On Functions Weakly Computable by Petri Nets and Vector Addition Systems*

J. Leroux[1] and Ph. Schnoebelen[2]

[1] LaBRI, Univ. Bordeaux & CNRS, France
[2] LSV, ENS Cachan & CNRS, France

Abstract. We show that any unbounded function weakly computable by a Petri net or a VASS cannot be sublinear. This answers a long-standing folklore conjecture about weakly computing the inverses of some fast-growing functions. The proof relies on a pumping lemma for sets of runs in Petri nets or VASSes.

1 Introduction

Petri nets (PN), Vector Addition Systems (VAS) and Vector Addition Systems with States (VASS) are essentially equivalent computational models based on simple operations on positive integer counters: decrements and increments. Such systems can be used to compute number-theoretical functions, exactly like with Minsky machines or Turing machines. However, they cannot compute all recursive functions since they are less expressive than Minsky machines. In particular they lack zero-tests, or, more precisely, they cannot initiate a given action on the condition that a counter's value is zero, only on the condition that it is not zero.

The standard definition for a function computed by a Petri net or a VASS is due to Rabin and is called "functions weakly computable by a Petri net", or just "WCPN functions" (all definitions will be found in the following sections). This notion has been used since the early days of Petri nets and has proved very useful in hardness or impossibility proofs: The undecidability of equivalence problems for nets and VASSes, and the Ackermann-hardness of the same problems for bounded systems, have been proved using the fact that multivariate polynomials with positive integer coefficients —aka positive Diophantine polynomials— and, respectively, the fast-growing functions $(F_i)_{i\in\mathbb{N}}$ in the Grzegorczyk hierarchy, are all WCPN [13,25,17].

The above results rely on showing how some useful functions are WCPN. But not much is known about exactly which functions are WCPN or not. It is known that all WCPN functions are monotonic. They are all primitive-recursive. The class of WCPN functions is closed under composition. A folklore conjecture states that the inverses of the fast-growing functions are not WCPN. It is stated as fact in [30, p.252] but no reference is given. In this paper, we settle the issue by proving that if $f : \mathbb{N} \to \mathbb{N}$ is WCPN and unbounded then it is in $\Omega(x)$, i.e., $f(x)$ eventually dominates $c \cdot x$ for some constant $c > 0$. Thus any function that is sublinear, like $x \mapsto \lfloor\sqrt{x}\rfloor$, or $x \mapsto \lfloor\log x\rfloor$, are not WCPN. In particular, this applies to the inverse F_i^{-1} of any fast-growing function with $i \geq 2$. The proof technique is interesting in its own right: it relies on a pumping lemma on sets of runs in VASSes or Petri nets.

* Work supported by the ReacHard project, ANR grant 11-BS02-001-01.

Beyond Petri nets and VASSes. Petri nets and VASSes are a classic example of well-structured systems [1,10]. In recent years, weakly computing numerical functions has proved to be a fundamental tool for understanding the expressive power and the complexity of some families of well-structured systems that are more powerful than Petri nets and VASSes [31,14,11]. For such systems, the hardness proofs rely on weakly computing fast-growing functions $(F_\alpha)_{\alpha\in Ord}$ that extend Grzegorczyk's hierarchy. These hardness proofs also crucially rely on weakly computing the inverses of the F_α's.

There are several extensions of Petri nets for which reachability (or coverability or termination) remains decidable: pushdown VASSes [23], nets with nested zero-tests [27], recursive VASSes [4] and Branching VASSes [6], VASSes with pointers to counters [5], etc. In many cases, it is not known how these extensions compare in expressive power and in complexity. We believe that weakly computable functions can be a useful tool when addressing theses questions.

Outline of the paper. Section 2 recalls the standard definitions for VASSes and fixes some notations. Section 3 recalls the definitions for WCPN functions and the classic results about them. Our main result is proved in Section 4.

2 Vector Addition Systems with States

Following Hopcroft and Pansiot [15], we adopt Vector Addition Systems with States (VASS) as our mathematical setting (rather than Petri nets or plain VASes) because they offers a good compromise between ease of description for specific systems, and convenient mathematical notation for reasoning about them. Nevertheless, all these models are essentially equivalent for our purposes.

Vectors of integers. $\mathbb{Z}$ and $\mathbb{N}$ denote the sets of integers and, resp., non-negative integers. For $d \in \mathbb{N}$, a d-dimensional *vector* is a tuple $\boldsymbol{a} = \langle a_1, \ldots, a_d\rangle$ in $\mathbb{Z}^d$. We use $\mathbf{0}$ to denote $\langle 0, \ldots, 0\rangle$ when the dimension is understood. Vectors can be concatenated: for $\boldsymbol{a} \in \mathbb{Z}^d$ and $\boldsymbol{b} \in \mathbb{Z}^{d'}$ we may write $\langle \boldsymbol{a}, \boldsymbol{b}\rangle$ for the vector $\langle a_1, \ldots, a_d, b_1, \ldots, b_{d'}\rangle \in \mathbb{N}^{d+d'}$.

Vectors in $\mathbb{Z}^d$ are ordered with $\boldsymbol{a} = \langle a_1, \ldots, a_d\rangle \sqsubseteq \boldsymbol{b} = \langle b_1, \ldots, b_d\rangle \overset{\text{def}}{\Leftrightarrow} a_1 \leq b_1 \wedge \cdots \wedge a_d \leq b_d$, and can be added with $(\boldsymbol{a} + \boldsymbol{b}) \overset{\text{def}}{=} \langle a_1 + b_1, \ldots, a_d + b_d\rangle$. Note that $(\mathbb{Z}^d, +, \mathbf{0})$ is a commutative monoid, having $(\mathbb{N}^d, +, \mathbf{0})$ as submonoid. Clearly, $\boldsymbol{a} \in \mathbb{N}^d$ iff $\mathbf{0} \sqsubseteq \boldsymbol{a}$. By Dickson's Lemma, $(\mathbb{N}^d, \sqsubseteq)$ is a well-ordering (more details in Section 4). In the following, we reserve $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \ldots$ for vectors in $\mathbb{N}^d$ and write $\|\boldsymbol{x}\|, \ldots$, for their norms, defined by $\|\langle x_1, \ldots, x_d\rangle\| \overset{\text{def}}{=} x_1 + \cdots + x_d$.

VASSes and their operational semantics. A VASS is a triple $\mathcal{A} = \langle d, Q, T\rangle$ where $d \in \mathbb{N}$ is a *dimension* (i.e., a number of counters), Q is a non-empty finite set of *(control) locations*, and $T \subseteq Q \times \mathbb{Z}^d \times Q$ is a finite set of *(transition) rules*. We usually write $q, q', \ldots$ for locations, and $t, \ldots$ for rules.

Fix some VASS $\mathcal{A} = \langle d, Q, T\rangle$. The operational semantics of $\mathcal{A}$ is given in the form of a transition system $(\mathit{Conf}, \rightarrow)$. Formally, $\mathit{Conf} \overset{\text{def}}{=} Q \times \mathbb{N}^d$ is the set of *configurations*, with typical elements $c, c', \ldots$ The labeled transition relation $\rightarrow \subseteq \mathit{Conf} \times \mathbb{Z}^d \times \mathit{Conf}$ is a set of triples $(c, \boldsymbol{a}, c')$ called *steps*. As is standard, we write $c \xrightarrow{\boldsymbol{a}} c'$ rather than

$(c, \boldsymbol{a}, c') \in \rightarrow$. Steps are defined by $(q, \boldsymbol{x}) \xrightarrow{\boldsymbol{a}} (q', \boldsymbol{y}) \overset{\text{def}}{\Leftrightarrow} (q, \boldsymbol{a}, q') \in T \wedge \boldsymbol{y} = \boldsymbol{x} + \boldsymbol{a}$. The vector $\boldsymbol{a}$ in a rule $(q, \boldsymbol{a}, q')$ is called a *translation*.

Graphical description. It is convenient to present VASSes in graphic form. See an example in Fig. 1. Here the locations and rules are depicted as a directed graph, as is standard with such automata-theoretical notions. Indeed, we see a d-dim VASS as being an automaton acting on d registers, or *counters*, capable of storing a natural number. These counters are named in our graphical depictions (see x and z in Fig. 1) so that we may use programming languages notations for the translation $\boldsymbol{a}$ in a rule $(q, \boldsymbol{a}, q')$. For example, in Fig. 1, the loop labeled with "x--;x--;z++" is a rule $(q_2, \boldsymbol{a}, q_2)$ with $\boldsymbol{a} = \langle -2, 1\rangle$, the other rule being $(q_1, \boldsymbol{0}, q_2)$.

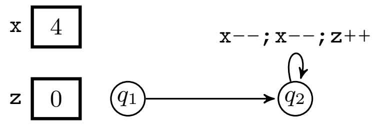

Fig. 1. A VASS depicted as an automaton acting on counters.

Runs and reachability. For $k \in \mathbb{N}$, a length-k *run* from c to c' is a sequence ρ of the form $c_0\, \boldsymbol{a}_1\, c_1 \cdots \boldsymbol{a}_k\, c_k$ that alternates between configurations and vectors and such that $c_0 = c$, $c_k = c'$, and $c_{i-1} \xrightarrow{\boldsymbol{a}_i} c_i$ for all $i = 1, \ldots, k$. For a run ρ as above, we let $\mathrm{src}(\rho)$ and $\mathrm{tgt}(\rho)$ denote c_0 and, respectively, c_k. We write $c \xrightarrow{*} c'$ when there is a run ρ from c to c', in which case we say that c' is *reachable* from c.

Continuing our previous example, the reachability relation for the VASS from Fig. 1 can be captured[3] with the following:

$$(q_i, x, z) \xrightarrow{*} (q_j, x', z') \quad \text{iff} \quad \begin{cases} 0 \le x - x' = 2(z' - z) & \text{if } j = 2, \text{ or} \\ x = x' \wedge z = z' & \text{if } i = j = 1. \end{cases} \tag{1}$$

Lifting steps and runs. It is well known and easy to see that steps can be lifted up by vectors $\boldsymbol{z} \in \mathbb{N}^d$. For a configuration $c = (q, \boldsymbol{x})$, we write $c + \boldsymbol{z}$ for the configuration $(q, \boldsymbol{x} + \boldsymbol{z})$. The following properties will be useful in later sections:

Lemma 2.1 (Lifting steps and runs). *For all $c, c' \in Conf$, $\boldsymbol{a} \in \mathbb{Z}^d$, and $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^d$:*

$$c \xrightarrow{\boldsymbol{a}} c' \textit{ implies } c + \boldsymbol{u} \xrightarrow{\boldsymbol{a}} c' + \boldsymbol{u}\,, \tag{2}$$

$$c \xrightarrow{*} c' \textit{ implies } c + \boldsymbol{u} \xrightarrow{*} c' + \boldsymbol{u}\,, \tag{3}$$

$$c + \boldsymbol{u} \xrightarrow{*} c + \boldsymbol{v} \textit{ implies } \forall x \in \mathbb{N} : c + x \cdot \boldsymbol{u} \xrightarrow{*} c + x \cdot \boldsymbol{v}\,. \tag{4}$$

Proof of (4). With $c + \boldsymbol{u} \xrightarrow{*} c + \boldsymbol{v}$ and Eq. (3) one obtains $c + \boldsymbol{u} + i \cdot \boldsymbol{u} + j \cdot \boldsymbol{v} \xrightarrow{*} c + i \cdot \boldsymbol{u} + \boldsymbol{v} + j \cdot \boldsymbol{v}$ for any $i, j \in \mathbb{N}$. Chaining such runs yields

$$c + x \cdot \boldsymbol{u} \xrightarrow{*} c + (x-1) \cdot \boldsymbol{u} + \boldsymbol{v} \xrightarrow{*} c + (x-2) \cdot \boldsymbol{u} + 2 \cdot \boldsymbol{v} \xrightarrow{*} \cdots \xrightarrow{*} c + x \cdot \boldsymbol{v}\,. \quad \square$$

[3] The "$\Rightarrow$" direction is proved by induction on the length of the run, where every additional step respects the invariant stated by Eq. 1. The "$\Leftarrow$" direction is obvious when $j = 1$, and proved by concatenating steps of the form $(q_2, x, z) \rightarrow (q_2, x-2, z+1)$ when $j = 2$.

3 Weakly Computable Functions

In this section we recall the classic notion of weak PN computers and weakly computable functions. We recall the main known results, most of them from the 70's or early 80's, when the applications were limited to a few hardness or impossibility proofs. This material is classic but has been partly forgotten.

As we argued in the introduction, the notion of weakly computable functions has recently gained new relevance with the development of well-structured systems that go beyond Petri nets and VASSes in expressive power, while sharing some of their characteristics. In particular, we expect that it will help understanding the expressive power of extensions like VASSes with nested zero-tests [27] or with a pushdown stack [23].

3.1 Weak PN Computers and Weakly Computable Functions

The expected way for a finite-state register machine $\mathcal{A}$ to compute a numerical function $f : \mathbb{N} \to \mathbb{N}$ is to start in some initial location with some input value n stored in a designated input counter and from that configuration eventually reach a final or accepting location with $f(n)$ in a designated output counter. In order for $\mathcal{A}$ to be correct, it should be impossible that it reaches its accepting location with a value differing from $f(n)$ in the output counter. In that case, we say that $\mathcal{A}$ *strongly computes* f. This notion of correctness is fine with Minsky machines but it is too strong for VASSes and does not lead to an interesting family of computable functions. In fact, Petri nets and VASSes are essentially nondeterministic devices, and the above notion of strongly computing some function does not accommodate nondeterminism nicely.

With this in mind, Rabin defined a notion of "weakly computing f" that combines the following two principles:

Completeness: For any $n \in \mathbb{N}$, there is a computation with input n and output $f(n)$;
Safety: Any computation from input n to some output r satisfies $r \leq f(n)$.

This leads to our first definition:

Definition 3.1 (Weak PN computers). *Let $f : \mathbb{N}^n \to \mathbb{N}^m$ be a total function. A* weak PN computer *for f is a d-dimensional VASS $\mathcal{A}$, with $d \geq n + m$ and two designated locations q_{init} and q_{final}, that satisfies the following two properties. Here we write ℓ for $d - n - m$, and we decompose vectors $\boldsymbol{w} \in \mathbb{N}^d$ as concatenations $\boldsymbol{w} = \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ where $\boldsymbol{x} \in \mathbb{N}^n$, $\boldsymbol{y} \in \mathbb{N}^\ell$ and $\boldsymbol{z} \in \mathbb{N}^m$.*

$$\forall \boldsymbol{x} : \exists \boldsymbol{x}', \boldsymbol{y}' : (q_{init}, \boldsymbol{x}, \mathbf{0}, \mathbf{0}) \xrightarrow{*} (q_{final}, \boldsymbol{x}', \boldsymbol{y}', f(\boldsymbol{x})) \ , \tag{CO}$$

$$\forall \boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y}', \boldsymbol{z}' : (q_{init}, \boldsymbol{x}, \mathbf{0}, \mathbf{0}) \xrightarrow{*} (q_{final}, \boldsymbol{x}', \boldsymbol{y}', \boldsymbol{z}') \textit{ implies } \boldsymbol{z}' \sqsubseteq f(\boldsymbol{x}) \ . \tag{SA}$$

We say that f is weakly computable*, or WCPN, if there is a weak PN computer for it.*

For convenience, Definition 3.1 assumes that the (n-dimensional) input is given in the first n counters of $\mathcal{A}$, and that the m-dimensional result is found in its last m counters. Note that $\mathcal{A}$ may use its ℓ extra counters for auxiliary calculations.

Example 3.2 (A weak computer for halving). The VASS from Fig. 1 is a weak computer for $f : x \mapsto \lfloor \frac{x}{2} \rfloor$. We just have to designate q_1 and q_2 as the required q_{init} and, resp., q_{final}. To show that (CO) and (SA) hold, one sets $z = 0$ in Eq. (1). This gives

$$(q_1, x, 0) \xrightarrow{*} (q_2, x', z') \text{ iff } z' = \frac{x - x'}{2} \,,$$

entailing both (CO) —pick $x' = (x \mod 2)$— and (SA) —since $z', x' \in \mathbb{N}$—. □

Only monotonic functions can be weakly computed in the above sense. This is an immediate consequence of the monotonicity of steps in VASSes (see Lemma 2.1).

Proposition 3.3 (Monotonicity of WCPN functions). *If $f : \mathbb{N}^n \to \mathbb{N}^m$ is WCPN then $\boldsymbol{x} \sqsubseteq \boldsymbol{x}'$ implies $f(\boldsymbol{x}) \sqsubseteq f(\boldsymbol{x}')$.*

Proof. Assume that $\boldsymbol{x} \sqsubseteq \boldsymbol{x}'$ and pick any weak PN computer for f. By (CO), there is a run $(q_{\text{init}}, \boldsymbol{x}, \mathbf{0}, \mathbf{0}) \xrightarrow{*} (q_{\text{final}}, \boldsymbol{v}, \boldsymbol{y}, f(\boldsymbol{x}))$. By Eq. (3), there is also a run $(q_{\text{init}}, \boldsymbol{x}', \mathbf{0}, \mathbf{0}) \xrightarrow{*} (q_{\text{final}}, \boldsymbol{v} + \boldsymbol{x}' - \boldsymbol{x}, \boldsymbol{y}, f(\boldsymbol{x}))$. Thus $f(\boldsymbol{x}) \sqsubseteq f(\boldsymbol{x}')$ by (SA). □

3.2 More weakly computable functions

Example 3.4 (A weak computer for multiplication, from [26]). Fig. 2 describes $\mathcal{A}_\times$, a weak computer for $f : x_1, x_2 \mapsto x_1 \times x_2$. To show that (SA) holds, we associate with

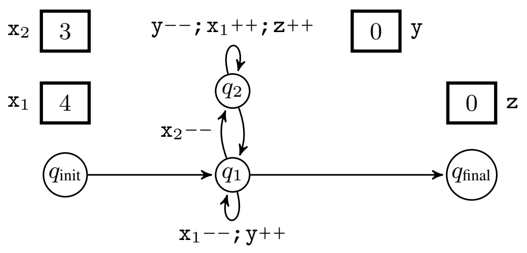

Fig. 2. $\mathcal{A}_\times$, a VASS weakly computing $x_1, x_2 \mapsto x_1 \times x_2$.

any configuration c of $\mathcal{A}_\times$ a value $M(c) \in \mathbb{N}$ given by

$$M(q, x_1, x_2, y, z) \stackrel{\text{def}}{=} \begin{cases} z + (x_1 + y) \cdot x_2 + y & \text{if } q = q_2, \\ z + (x_1 + y) \cdot x_2 & \text{otherwise.} \end{cases} \tag{5}$$

By considering all rules of $\mathcal{A}_\times$ in turn, one checks that $c \to c'$ implies $M(c) \geq M(c')$. Thus given an arbitrary run from $c_0 = (q_{\text{init}}, x_1, x_2, 0, 0)$ to $c_k = (q_{\text{final}}, x'_1, x'_2, y', z')$ it holds that $M(c_0) \geq M(c_k)$, i.e., $x_1 \cdot x_2 \geq z' + x'_1 \cdot x'_2 + y' \cdot x'_2$. This entails $z' \leq x_1 \cdot x_2$ as required by (SA).

We let the reader check that (CO) holds. [Hint: steps $c \to c'$ that only use the rule from q_1 to q_2 when $\mathtt{x}_1 = 0$, and from q_2 to q_1 when $\mathtt{y} = 0$, satisfy $M(c) = M(c')$.] □

Weakly computing functions has mainly been used in hardness or impossibility proofs. For example, weakly computing multiplication can be used to show that reachability sets are not always semilinear: it is easy to adapt the construction underlying $\mathcal{A}_\times$ and design a VASS that, starting from a fixed c_0, generates the set of triples $\{\langle y_1, y_2, y\rangle \in \mathbb{N}^3 \mid 0 \leq y \leq y_1 \cdot y_2\}$ in some designated counters. The reachability set of this VASS cannot be semilinear.

Proposition 3.5. *The class of WCPN functions is closed by composition.*

Proof (Idea). The obvious way of gluing a weak PN computer for g after a weak PN computer for f produces a weak PN computer for $g \circ f$. To prove that the resulting VASS satisfies (SA), one observes that any run can be reordered by firing all rules in the f part before the rules in the g part. $\square$

Since the class of WCPN functions contains addition, multiplication, projections, and tuplings, one deduces that all positive Diophantine polynomials (multivariate polynomials with coefficients in $\mathbb{N}$) are weakly computable. This was used by Rabin in his reduction of Hilbert's 10th Problem to the inclusion problem for VASS reachability sets [3,13]. Hack strengthened this reduction to show that already the equality problem was undecidable [13]. (Later, Jančar showed that all behavioural equivalences are undecidable for VASSes —already for dimension $d = 5$—, using a simpler reduction with some notion of weak computer that is not numerical [18].)

3.3 Iterable weak PN computers

There are other easy and useful examples of WCPN functions that are not positive Diophantine polynomials, like $\min$ and $\max$, or even *half* seen previously. In order to show the weak computability of more functions, in particular functions that are not polynomially or exponentially bounded, Mayr [24] introduced the following notion:

Definition 3.6 (Iterable Weak PN Computers). *Let $f : \mathbb{N} \to \mathbb{N}$ be a weakly computable unary function. A weak PN computer $\mathcal{A}$ for f is* iterable *if it satisfies*

$$\forall \boldsymbol{w}, \boldsymbol{w}' : (q_{\mathit{init}}, \boldsymbol{w}) \xrightarrow{*} (q_{\mathit{final}}, \boldsymbol{w}') \textit{ implies } \|\boldsymbol{w}'\| \leq f(\|\boldsymbol{w}\|) . \tag{IT}$$

A unary function is iterably weakly computable*, or IWCPN, if there exists an iterable weak PN computer for it.*

In Definition 3.6, $\|\boldsymbol{w}\|$ counts all the tokens (using Petri net terminology) in the starting configuration. The property stated by Eq. (IT) is useful in constructions that include $\mathcal{A}$ and where one cannot guarantee that all computations by $\mathcal{A}$ will start from clean configurations with zeroes in $\boldsymbol{y}$ and $\boldsymbol{z}$.

Example 3.7 (Halving). The weak PN computer for halving (Fig. 1) is *not iterable* since, by Eq. (1), it has runs like $(q_{\text{init}}, 0, 1) \xrightarrow{*} (q_{\text{final}}, 0, 1)$ and $(q_{\text{init}}, 1, 0) \xrightarrow{*} (q_{\text{final}}, 1, 0)$ that have $\|\boldsymbol{w}\| = \|\boldsymbol{w}'\| = 1$, hence $\|\boldsymbol{w}'\| \not\leq f(\|\boldsymbol{w}\|) = \lfloor \frac{1}{2} \rfloor = 0$. $\square$

In fact, it is impossible to design an iterable weak PN computer for halving, as a consequence of the following proposition.

Proposition 3.8 (Strict Monotonicity of IWCPN functions). *If $f : \mathbb{N} \to \mathbb{N}$ is IWCPN then $f(x) < f(x+1)$ for all $x \in \mathbb{N}$.*

Proof. Assume a given iterable weak PN computer for f and let $x \in \mathbb{N}$. From (CO), we derive $(q_{\text{init}}, x, \mathbf{0}, 0) \xrightarrow{*} (q_{\text{final}}, x', \boldsymbol{y}', f(x))$. By lifting, we get $(q_{\text{init}}, x+1, \mathbf{0}, 0) \xrightarrow{*} (q_{\text{final}}, x'+1, \boldsymbol{y}', f(x))$. Now (IT) gives $x'+1+\|\boldsymbol{y}'\|+f(x) \leq f(x+1)$, which entails $f(x) < f(x+1)$. □

Example 3.9 (Doubling is IWCPN). One may design an iterable weak computer for

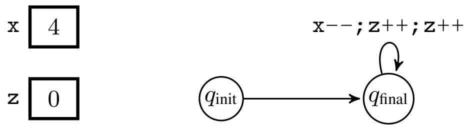

Fig. 3. $\mathcal{A}_{dbl}$, a VASS weakly computing $x \mapsto 2 \cdot x$.

$x \mapsto 2 \cdot x$ (doubling) by a slight modification of Fig. 1. The resulting VASS, called $\mathcal{A}_{dbl}$, is depicted in Fig. 3. It satisfies

$$(q_{\text{init}}, x, z) \xrightarrow{*} (q_{\text{final}}, x', z') \quad \text{iff} \quad 0 \leq 2(x - x') = z' - z\,.$$

These runs have thus $\|\boldsymbol{w}'\| = x' + z' = 2x - x' + z$. On the other hand $f(\|\boldsymbol{w}\|) = f(x+z) = 2x + 2z$. Hence $\|\boldsymbol{w}'\| \leq f(\|\boldsymbol{w}'\|)$ as required by (IT). □

As expected, the functions weakly computed by iterable weak PN computers are iterable. Given a unary f and some $n \in \mathbb{N}$, we write $f^n(x)$ for the n-fold application $f(f(f(\cdots(x)\cdots)))$ of f. In particular $f^0(x) = x$. One can then show the following:

Proposition 3.10 ([24,26]). *If f is IWCPN, then $iter(f) : x, y \mapsto f^x(y)$ is IWCPN.*

Indeed, the whole point of (IT) is to entail the correctness of the obvious construction for iterating a weak PN computer.

With Proposition 3.10, and since doubling is IWCPN, we deduce that $iter(dbl)$: $x, y \mapsto dbl^x(y) = 2^x y$ is IWCPN. From that we deduce that $tower : x \mapsto 2^{2^{\cdot^{\cdot^{2}}}}\}x$ times is IWCPN. Continuing, all the fast-growing functions $(F_i)_{i\in\mathbb{N}}$ in the Grzegorczyk hierarchy are IWCPN. This was used by Mayr to show that the inclusion problem for finite reachability sets is not primitive recursive [24,25]. The problem is in fact $\mathbf{F}_\omega$-complete in the recent classification of Schmitz [28]. Using the same IWCPN functions, Jančar showed that all behavioural equivalences are Ackermann-hard between bounded VASSes [17].

While the fast-growing hierarchy extends beyond the $(F_i)_{i\in\mathbb{N}}$, the functions at the higher levels —starting with F_ω which is one possible form for Ackermann's function— are not WCPN. The following Proposition is folklore, but we could not find it explicitly stated in the literature:

Proposition 3.11. *Any weakly computable function f is primitive recursive.*

Proof. For a VASS $\mathcal{A}$ and a starting configuration $c_0 \in \mathit{Conf}$, let $S_{\mathcal{A}}(c_0) \in \mathbb{N}$ be the maximum norm of a configuration occurring in the Karp-Miller tree $T_{\mathit{KM}}(c_0)$ rooted in c_0. (We assume familiarity with Karp-Miller trees, otherwise see [19, Section 4A]. Note that these trees really contain "extended configurations" in $Q \times (\mathbb{N} \cup \{\omega\})^d$ but one only uses the finite values for their norm, as in, e.g., $\|(q, 7, \omega, 2)\| = 7 + 2 = 9$.)

For $k \in \mathbb{N}$, let now $S_{\mathcal{A}}(k) \stackrel{\text{def}}{=} \max\{S_{\mathcal{A}}(c) \mid \|c\| \leq k\}$, i.e., $S_{\mathcal{A}}(k)$ is the size of the largest configuration in a Karp-Miller tree for $\mathcal{A}$ starting in some initial configuration of size at most k. It is shown in [9, Section 7C] that $S_{\mathcal{A}} : \mathbb{N} \to \mathbb{N}$ is primitive recursive (using a different norm for vectors, but this is of no consequence here). If now $\mathcal{A}$ is a weak PN computer for some $f : \mathbb{N}^n \to \mathbb{N}^m$, then by (CO) and for any $\boldsymbol{x} \in \mathbb{N}^n$, the tree $T_{\mathit{KM}}((q_{\text{init}}, \boldsymbol{x}, \mathbf{0}, \mathbf{0}))$ contains an extended configuration $c_{\boldsymbol{x}} = (q_{\text{final}}, \boldsymbol{x}', \boldsymbol{y}', \boldsymbol{z}')$ that covers $(q_{\text{final}}, \mathbf{0}, \mathbf{0}, f(\boldsymbol{x}))$ since every reachable configuration is covered in T_{KM}. Furthermore, by (SA), no configuration reachable in q_{final} can have values above $f(\boldsymbol{x})$ in the last m counters. Hence the $\boldsymbol{z}'$ part of $c_{\boldsymbol{x}}$ has no ω's and $\boldsymbol{z}' = f(\boldsymbol{x})$.

Finally, one can compute $f(\boldsymbol{x})$ by building a Karp-Miller tree of size that is primitive recursive in $\|\boldsymbol{x}\|$ and by reading $f(\boldsymbol{x})$ on one of its leaves. $\square$

3.4 Alternative Definitions

The literature contains other proposals for a notion of weakly computable functions, all of them based on Rabin's seminal idea. One may define the function weakly computed by $\mathcal{A}$ as the maximum number of times a given transition [13], or all transitions [12], can be fired between $(q_{\text{init}}, \boldsymbol{x}, \mathbf{0})$ and q_{final}. While this does not give a larger class of weakly computable functions, other proposals are richer as we now show.

Weakly Computing Eagerly. An interesting notion is that of "eagerly" weakly computable functions. For this, we modify the Correctness and Safety requirement in Definition 3.1, replacing them with

$$\forall \boldsymbol{x} : (q_{\text{init}}, \boldsymbol{x}, \mathbf{0}, \mathbf{0}) \xrightarrow{*} (q_{\text{final}}, \mathbf{0}, \mathbf{0}, f(\boldsymbol{x}))\,, \qquad \text{(CO')}$$

$$\forall \boldsymbol{x}, \boldsymbol{z}' : (q_{\text{init}}, \boldsymbol{x}, \mathbf{0}, \mathbf{0}) \xrightarrow{*} (q_{\text{final}}, \mathbf{0}, \mathbf{0}, \boldsymbol{z}') \text{ implies } \boldsymbol{z}' \sqsubseteq f(\boldsymbol{x})\,, \qquad \text{(SA')}$$

and we then say that f is EWCPN.

The idea here is that $\mathcal{A}$ must have consumed inputs and auxiliary counters *at the end of the computation*. This is meaningful in some reductions —e.g., reducing reachability in VASSes to some problem on some weakly computable function—. To the best of our knowledge, it has never been considered in the literature on VASSes and Petri nets. The fact is that it does not behave as nicely as the classical definition (see below). However, it is a natural option with some extensions like VASSes extended with resets as in [2,8], or with nested zero-tests as in [27].

Fact 3.12. *The class of EWCPN functions strictly extends the WCPN functions.*

Proof. Obviously, any WCPN function can be computed eagerly: a weak PN computer for f is made eager by adding decrementing rules that can empty the input and auxiliary counters. To see that the extension is strict, note that EWCPN functions are not always monotonic. For example, $\mathit{parity} : x \mapsto x \mod 2$ is easily seen to be EWCPN. $\square$

Since EWCPN functions are not necessarily monotonic, it is not clear whether the composition of two EWCPN functions is EWCPN itself. We introduced this notion because our main result (Theorem 4.7) holds for the larger class of EWCPN functions.

Families of Weak PN Computers. Very often, reductions showing hardness do not need a *fixed* weak PN computer for some f. They can accommodate a family $(\mathcal{A}_i)_{i\in\mathbb{N}}$ such that each $\mathcal{A}_i$ weakly computes $f(i)$, or $f(x)$ for all $x = 0, \ldots, i$. The family needs to be simple in computational terms —typically "*polynomial-time uniform*"— so that it can be used in reductions. Since the $\mathcal{A}_i$'s that weakly compute the fast-growing F_i's are uniformly generated, they provide a family weakly computing the function F_ω (equivalently, Ackermann's function) defined by $F_\omega(x) = F_x(x)$.

For slow-growing functions, one often needs a family that is polynomial-time uniform *in the size of* $f(x)$. A recent example is Lazić's polynomial-time uniform family of pushdown VASSes (VASSes extended with a stack) that weakly computes the inverse of $tower : x \mapsto 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\}x \text{ times}$ [21].

4 Well-Quasi-Ordering Runs in VASSes

Recall that a *quasi-order* (qo) on a set S is a binary relation $\preceq$ on S that is reflexive and transitive. A *partial order* is an antisymmetric quasi-order. A *well-quasi-order* (wqo) on S is a quasi-order $\preceq$ such that every infinite sequence $s_0, s_1, s_2, \ldots$ in S contains an increasing pair $s_i \preceq s_j$ for some $i < j$. See [20] or [29] for more on wqos.

Example 4.1. It is well-known that $(\mathbb{N}, \leq)$ is a wqo, while $(\mathbb{Z}, \leq)$ or $(\mathbb{Q}_{\geq 0}, \leq)$ are not. As another example, the pigeonhole principle shows that the partial order $(S, =)$ is a wqo if, and only if, S is finite. □

In practice, many wqos are defined by applying well-known constructions on standard, already known, wqos.

Definition 4.2 (Products of wqos and Dickson's Lemma). *Let $(S_1, \preceq_1), \ldots, (S_n, \preceq_n)$ be qos. Their product is the qo $(S_\times, \preceq_\times)$ with $S_\times \overset{\text{def}}{=} S_1 \times \cdots \times S_n$ and $\preceq_\times$ given by*

$$(s_1, \ldots, s_n) \preceq_\times (s'_1, \ldots, s'_n) \overset{\text{def}}{\Leftrightarrow} s_1 \preceq_1 s'_1 \wedge \ldots \wedge s_n \preceq_n s'_n \;.$$

It is well-known that $(S_\times, \preceq_\times)$ is a wqo if all $(S_i, \preceq_i)$ are.

This last result is standardly called Dickson's Lemma, after Dickson's proof of his "Lemma A" in [7], showing in essence that any subset of $\mathbb{N}^d$ has finitely many minimal elements. For our purpose, Dickson's Lemma shows that $(\mathbb{N}^d, \sqsubseteq)$ is a wqo.

Definition 4.3 (Sequence extensions of wqos and Higman's Lemma). *The sequence extension $(S^*, \preceq_*)$ of a qo $(S, \preceq)$ has for its support the set S^* of all finite sequences $s_1 \cdots s_k$ over S, and these sequences are ordered by*

$$s_1 \cdots s_k \preceq_* s'_1 \cdots s'_\ell \overset{\text{def}}{\Leftrightarrow} \begin{cases} \text{there are indexes } 1 \leq n_1 < \cdots < n_k \leq \ell \\ \text{such that } s_1 \preceq s'_{n_1} \wedge \cdots \wedge s_k \preceq s'_{n_k}. \end{cases}$$

It is well-known that $(S^*, \preceq_*)$ is a wqo if $(S, \preceq)$ is.

$$\begin{tikzcd}
 & \rho_0 & & & \rho_1 & & \rho_2 \\
\rho' \arrow[d, dotted, no head, "\trianglerighteq"] & (q_1,3,3) \arrow[r, "\boldsymbol{b}"] \arrow[dr, dotted, no head, "\sqsupseteq"] & (q_1,2,1) \arrow[r, "\boldsymbol{a}_1"] \arrow[d, dotted, no head, "\sqsupseteq"] & (q_2,3,2) \arrow[r, "\boldsymbol{b}'"] \arrow[dr, dotted, no head, "\sqsupseteq"] & (q_3,2,0) \arrow[r, "\boldsymbol{b}''"] & (q_2,2,2) \arrow[r, "\boldsymbol{a}_2"] \arrow[dl, dotted, no head, "\sqsupseteq"] & (q_3,0,2) \arrow[d, dotted, no head, "\sqsupseteq"] \\
\rho & & (q_1,1,0) \arrow[rr, "\boldsymbol{a}_1"] & & (q_2,2,1) \arrow[rr, "\boldsymbol{a}_2"] & & (q_3,0,1)
\end{tikzcd}$$

Fig. 4. Example for Def. 4.4: A factorization of $\rho' = \rho_0\boldsymbol{a}_1\rho_1\boldsymbol{a}_2\rho_2$ witnessing $\rho \trianglelefteq \rho'$.

This last result is called Higman's Lemma.

In the rest of this section, we assume a fixed VASS $\mathcal{A} = (d, Q, T)$. We now define three orderings, respectively between the configurations of $\mathcal{A}$, between its steps, and between its runs, in the following way:

$$(q_1, \boldsymbol{x}_1) \sqsubseteq (q_2, \boldsymbol{x}_2) \overset{\text{def}}{\Leftrightarrow} q_1 = q_2 \wedge \boldsymbol{x}_1 \sqsubseteq \boldsymbol{x}_2\ , \tag{6}$$

$$(c_1 \xrightarrow{\boldsymbol{a}_1} c_1') \preccurlyeq (c_2 \xrightarrow{\boldsymbol{a}_2} c_2') \overset{\text{def}}{\Leftrightarrow} c_1 \sqsubseteq c_2 \wedge \boldsymbol{a}_1 = \boldsymbol{a}_2 \wedge c_1' \sqsubseteq c_2'\ . \tag{7}$$

Since Q and T are finite, $(Q, =)$ and $(T, =)$ are wqos, hence $(\mathit{Conf}, \sqsubseteq)$ and $(\mathit{Conf} \times T \times \mathit{Conf}, \preccurlyeq)$ are wqos by Dickson's Lemma.

Definition 4.4 (Ordering runs, see Fig. 4). *For two runs ρ, ρ' of $\mathcal{A}$, we write $\rho \trianglelefteq \rho'$ if $\rho = c_0\boldsymbol{a}_1c_1 \ldots \boldsymbol{a}_kc_k$ and ρ' can be factored as some $\rho' = \rho_0\boldsymbol{a}_1\rho_1 \ldots \boldsymbol{a}_k\rho_k$ where, for all $j = 0, \ldots, k$, the ρ_j factor is a run such that $c_j \sqsubseteq \mathrm{src}(\rho_j)$ and $c_j \sqsubseteq \mathrm{tgt}(\rho_j)$.*

Lemma 4.5. *The relation $\trianglelefteq$ is a wqo over the runs of $\mathcal{A}$.*

Proof. This is essentially [22, Lemma 4.1] or [16, Theorem 6.5]. A more direct proof is by observing that

$$\rho \trianglelefteq \rho' \text{ iff} \begin{cases} \mathrm{src}(\rho) \sqsubseteq \mathrm{src}(\rho') \ \wedge\ \mathrm{tgt}(\rho) \sqsubseteq \mathrm{tgt}(\rho') \ \wedge \\ (c_0 \xrightarrow{\boldsymbol{a}_1} c_1)(c_1 \xrightarrow{\boldsymbol{a}_2} c_2) \cdots (c_{k-1} \xrightarrow{\boldsymbol{a}_k} c_k) \preccurlyeq_* (c_0' \xrightarrow{\boldsymbol{a}_1'} c_1') \cdots (c_{\ell-1}' \xrightarrow{\boldsymbol{a}_\ell'} c_\ell')\ , \end{cases}$$

where $\preccurlyeq_*$ is the sequence extension of the ordering of steps defined with Eq. (7). Since $\preccurlyeq$ (over steps) and $\sqsubseteq$ (over configurations) are wqos, $\trianglelefteq$ is a wqo over the runs of $\mathcal{A}$. $\square$

The ordering on runs comes with the following Pumping Lemma:

Lemma 4.6 (Pumping Lemma). *Let $\rho \trianglelefteq \rho'$ and let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^d$ such that* $\mathrm{src}(\rho') = \mathrm{src}(\rho) + \boldsymbol{u}$ *and* $\mathrm{tgt}(\rho') = \mathrm{tgt}(\rho) + \boldsymbol{v}$. *Then*

$$\forall x \in \mathbb{N} : \big(\mathrm{src}(\rho) + x \cdot \boldsymbol{u}\big) \ \xrightarrow{*}\ \big(\mathrm{tgt}(\rho) + x \cdot \boldsymbol{v}\big)\ .$$

Proof. Assume that ρ is some $c_0 \xrightarrow{\boldsymbol{a}_1} c_1 \cdots \xrightarrow{\boldsymbol{a}_k} c_k$ and that $\rho \trianglelefteq \rho'$ is witnessed by factoring ρ' under the form $\rho_0 \boldsymbol{a}_1 \rho_1 \ldots \boldsymbol{a}_k \rho_k$. For $j = 0, \ldots, k$, write $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$ for the vectors in $\mathbb{N}^d$ such that $\mathrm{src}(\rho_j) = c_j + \boldsymbol{u}_j$ and $\mathrm{tgt}(\rho_j) = c_j + \boldsymbol{v}_j$. Observe that $\boldsymbol{u}_j = \boldsymbol{v}_{j-1}$ for any $j > 0$ since there are steps $c_{j-1} \xrightarrow{\boldsymbol{a}_j} c_j$ and $c_{j-1} + \boldsymbol{v}_{j-1} = \mathrm{tgt}(\rho_{j-1}) \xrightarrow{\boldsymbol{a}_j} \mathrm{src}(\rho_j) = c_j + \boldsymbol{u}_j$.

Since there is a run ρ_j of the form $c_j + \boldsymbol{u}_j \xrightarrow{*} c_j + \boldsymbol{v}_j$, by Lemma 2.1 there is also $c_j + x \cdot \boldsymbol{u}_j \xrightarrow{*} c_j + x \cdot \boldsymbol{v}_j$ for any $x \in \mathbb{N}$. We may now insert these runs between steps $c_{j-1} + x \cdot \boldsymbol{v}_{j-1} \xrightarrow{\boldsymbol{a}_j} c_j + x \cdot \boldsymbol{v}_{j-1} = c_j + x \cdot \boldsymbol{u}_j$, obtained by lifting up $c_{j-1} \xrightarrow{\boldsymbol{a}_j} c_j$, see Eq. (2). This gives $c_0 + x \cdot \boldsymbol{u} \xrightarrow{*} c_k + x \cdot \boldsymbol{v}$ as required. □

(Alternatively, it is shown in [16, Lemma 6.7] that if $\rho \trianglelefteq \rho' \trianglelefteq \rho_x$ for some run ρ_x, then there exists a run ρ_{x+1} such that $\rho_x \trianglelefteq \rho_{x+1}$, $\mathrm{src}(\rho_{x+1}) = \mathrm{src}(\rho_x) + \boldsymbol{u}$, and $\mathrm{tgt}(\rho_{x+1}) = \mathrm{tgt}(\rho_x) + \boldsymbol{v}$. Using induction over $x \in \mathbb{N}$, and starting with $\rho_0 = \rho$ and $\rho_1 = \rho'$, one can provide for any x a run ρ_x witnessing $\mathrm{src}(\rho) + x \cdot \boldsymbol{u} \xrightarrow{*} \mathrm{tgt}(\rho) + x \cdot \boldsymbol{v}$.)

Theorem 4.7. *Let $f : \mathbb{N} \to \mathbb{N}$ be an unbounded unary EWCPN function. Then there exist $r, s \in \mathbb{N}$ with $s > 0$ and such that $f(r + s \cdot x)$ is in $\Omega(x)$.*

Proof. Fix an eager weak PN computer $\mathcal{A}$ for f. For every $r \in \mathbb{N}$, $\mathcal{A}$ has a run ρ_r of the form $(q_{\text{init}}, r, \mathbf{0}, 0) \xrightarrow{*} (q_{\text{final}}, 0, \mathbf{0}, f(r))$. Since f is unbounded, there exists an infinite subset $R \subseteq \mathbb{N}$ such that $(f(r))_{r \in R}$ is strictly increasing. Since the runs are well-ordered by $\trianglelefteq$ (Lemma 4.5) there exists two indexes $r < r'$ in R such that $\rho_r \trianglelefteq \rho_{r'}$. By introducing $s = r' - r$, Lemma 4.6 shows that for every $x \in \mathbb{N}$, we have

$$\big(q_{\text{init}}, r + s \cdot x, \mathbf{0}, 0\big) \xrightarrow{*} \big(q_{\text{final}}, 0, \mathbf{0}, f(r) + x \cdot \big[f(r') - f(r)\big]\big) .$$

With (SA'), we deduce that $f(r + s \cdot x) \geq f(r) + x \cdot [f(r') - f(r)]$ for every $x \in \mathbb{N}$. The lemma follows from $f(r') - f(r) > 0$. □

Corollary 4.8. *Let $f : \mathbb{N} \to \mathbb{N}$ be an unbounded unary WCPN function. Then $f(x)$ is in $\Omega(x)$.*

Proof. Direct from Theorem 4.7 since, by Proposition 3.3, f is non-decreasing. □

Thus any sublinear function like $x \mapsto \lceil \sqrt{x} \rceil$ or $x \mapsto \lceil \log x \rceil$ is *not weakly computable* even in the eager sense. (We note that any monotonic bounded function is WCPN, e.g., as a $\max$ of finitely many threshold functions of the form "$\mathtt{x} \mapsto$ if $\mathtt{u} \sqsubseteq \mathtt{x}$ then $\mathtt{v}_{\text{hi}}$ else $\mathtt{v}_{\text{lo}}$" with $\mathtt{v}_{\text{lo}} \sqsubseteq \mathtt{v}_{\text{hi}}$.)

Corollary 4.8 can be extended beyond unary functions as follows. A function $f : \mathbb{N}^n \to \mathbb{N}$ is said to be unbounded on a component i, with $1 \leq i \leq n$, if, for some natural numbers $r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_n \in \mathbb{N}$, the following unary function is unbounded:

$$x \mapsto f(r_1, \ldots, r_{i-1}, x, r_{i+1}, \ldots, r_n).$$

For example, $(x_1, x_2) \mapsto x_1$ is a WCPN function that is unbounded on the first component. From Corollary 4.8, and the monotonicity property given by Proposition 3.3, it follows that every WCPN function $f : \mathbb{N}^n \to \mathbb{N}$ has $f(x_1, \ldots, x_n)$ in $\Omega(\max_{i \in I} x_i)$, where I is the set of unbounded components for f.

5 Concluding Remarks

We proved that Petri nets and VASSes cannot weakly compute numerical functions that are sublinear. This was a folklore conjecture that, to the best of our knowledge, had not yet been settled.

Traditionally, weakly computable functions have been used to prove hardness results. Recent hardness proofs for well-structured systems crucially rely on the ability to weakly compute both fast-growing *and slow-growing* functions. For Petri nets and VASSes, a weak computer for some slow-growing function could perhaps have been used (depending on its structure) to improve the currently best lower bound for the reachability problem. It was thus important to check whether such weak commputer exist.

Our negative result raises the question of whether more functions can be weakly computed in extensions of VASSes like the pushdown VASSes of [23] or the nets with nested zero-tests of [27].

References

1. P. A. Abdulla, K. Čerāns, B. Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information & Computation*, 160(1–2):109–127, 2000.
2. T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3(1):85–104, 1976.
3. H. G. Baker Jr. Rabin's proof of the undecidability of the reachability set inclusion problem of vector addition systems. Memo 79, Computation Structures Group, Project MAC, M.I.T., July 1973.
4. A. Bouajjani and M. Emmi. Analysis of recursively parallel programs. In *POPL 2012*, pages 203–214. ACM, 2012.
5. S. Demri, D. Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. In *LICS 2013*, pages 33–42. IEEE, 2013.
6. S. Demri, M. Jurdziński, O. Lachish, and R. Lazić. The covering and boundedness problems for branching vector addition systems. *Journal of Computer and System Sciences*, 79(1):23–38, 2013.
7. L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Amer. Journal Math.*, 35:413–422, 1913.
8. C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *ICALP '98*, volume 1443 of *Lect. Notes Comp. Sci.*, pages 103–115. Springer, 1998.
9. D. Figueira, S. Figueira, S. Schmitz, and Ph. Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's Lemma. In *LICS 2011*, pages 269–278. IEEE, 2011.
10. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.
11. Ch. Haase, S. Schmitz, and Ph. Schnoebelen. The power of priority channel systems. In *CONCUR 2013*, volume 8052 of *Lect. Notes Comp. Sci.*, pages 319–333. Springer, 2013. (Long version as CoRR Report 1301.5500 [cs.LO]).
12. M. Hack. *Decidability Questions for Petri Nets*. PhD thesis, Massachusetts Institute of Technology, June 1976. Available as report MIT/LCS/TR-161.
13. M. Hack. The equality problem for vector addition systems is undecidable. *Theoretical Computer Science*, 2(1):77–95, 1976.

14. S. Haddad, S. Schmitz, and Ph. Schnoebelen. The ordinal-recursive complexity of timed-arc Petri nets, data nets, and other enriched nets. In *LICS 2012*, pages 355–364. IEEE, 2012.
15. J. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979.
16. P. Jančar. Decidability of a temporal logic problem for Petri nets. *Theoretical Computer Science*, 74(1):71–93, 1990.
17. P. Jančar. Nonprimitive recursive complexity and undecidability for Petri net equivalences. *Theoretical Computer Science*, 256(1–2):23–30, 2001.
18. P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995.
19. R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.
20. J. B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory, Series A*, 13(3):297–305, 1972.
21. R. Lazić. The reachability problem for vector addition systems with a stack is not elementary. *CoRR*, abs/1310.1767, 2013.
22. J. Leroux. Vector addition systems reachability problem (a simpler solution). In *The Alan Turing Centenary Conference (Turing-100)*, volume 10 of *EasyChair Proceedings in Computing*, pages 214–228. EasyChair, 2012.
23. J. Leroux, M. Praveen, and G. Sutre. Hyper-Ackermannian bounds for pushdown vector addition systems. In *CSL-LICS 2014*. ACM, 2014.
24. E. W. Mayr. The complexity of the finite containment problem for Petri nets. Master's thesis, Massachusetts Institute of Technology, June 1977. Available as report MIT/LCS/TR-181.
25. E. W. Mayr and A. R. Meyer. The complexity of the finite containment problem for Petri nets. *Journal of the ACM*, 28(3):561–576, 1981.
26. H. Müller. Weak Petri net computers for Ackermann functions. *Elektronische Informationsverarbeitung und Kybernetik*, 21(4–5):236–246, 1985.
27. K. Reinhardt. Reachability in Petri nets with inhibitor arcs. *Electr. Notes Theor. Comput. Sci.*, 223:239–264, 2008.
28. S. Schmitz. Complexity hierarchies beyond elementary. Research Report 1312.5686 [cs.CC], Computing Research Repository, December 2013.
29. S. Schmitz and Ph. Schnoebelen. Algorithmic aspects of WQO theory. Lecture notes, 2012.
30. Ph. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.
31. Ph. Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *MFCS 2010*, volume 6281 of *Lect. Notes Comp. Sci.*, pages 616–628. Springer, 2010.