PVASS Reachability Is Decidable

Roland Guttenberg
University of Warsaw, Poland

Eren Keskin
TU Braunschweig, Germany

Roland Meyer
TU Braunschweig, Germany

Abstract

Reachability in pushdown vector addition systems with states (PVASS) is among the longest standing open problems in Theoretical Computer Science. We show that the problem is decidable in full generality. Our decision procedure is similar in spirit to the KLMST algorithm for VASS reachability, but works over objects that support an elaborate form of procedure summarization as known from pushdown reachability.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Pushdowns, Petri Nets, VASS, PVASS, Reachability, Decidability, Complexity

Digital Object Identifier 10.4230/LIPIcs.LICS.2026.53

Related Version *Full Version*: `https://arxiv.org/abs/2504.05015` [20]

Funding *Roland Guttenberg*: Supported by the ERC grant INFSYS, agreement no. 950398.

1 Introduction

The most basic models of computation beyond finite automata are pushdowns (manipulating a stack) and vector addition systems with states (manipulating a finite set of non-negative counters). Despite their simplicity, the interaction between the two models is not understood. Given a pushdown P and a VASS V, it is not known whether $L(P) \cap L(V) = \emptyset$ is decidable. We prove here that this is the case. Our algorithm has a complexity of Hyper-Ackermann, which is conjectured to be optimal [11].

The problem is most well-known in the community in its equivalent formulation, as the reachability problem in pushdown vector addition systems with states (PVASS). A PVASS is a finite automaton manipulating both a stack and non-negative counters. To solve PVASS reachability, one could imagine that simply combining the procedure summaries for pushdown reachability [50, 45] with the KLMST decomposition for VASS reachability [47, 41, 27, 28] would be enough. However, despite a long list of efforts over the last two decades, the problem could only be solved in special cases, and all attempts to generalize the solutions have failed [44, 2, 19, 40, 35, 3, 4, 5]. The combination requires new techniques.

Techniques. The first idea is to generalize the PVASS reachability problem in an appropriate fashion. We study the emptiness problem for a new model called *nested grammar vector addition systems* (NGVAS). NGVAS are nested context-free grammars: at level 0 terminals are counter updates in $\mathbb{Z}^d$, at level $n+1$ terminals are NGVAS of level n. NGVAS generalize PVASS via the standard conversion from pushdown automata to context-free grammars. The nesting allows us to approach algorithmic problems we encounter while solving reachability using induction.

The second idea is to prove a version of the Steinitz lemma [51, 49] for NGVAS, which will play an important role in a sufficient condition for reachability. The Steinitz lemma will be a corollary of a new result for context-free grammars that we call the *wide tree*

theorem and that we believe is of independent interest. Simply put, if G is non-linear and strongly-connected, then for many words $w \in L(G)$, there is a permutation $\pi(w) \in L(G)$ that has a derivation tree of *height logarithmic* in the length of w.

The third and final idea takes up most of the paper. To apply the aforementioned sufficient condition, we need pumpability properties. Despite similarities at first glance, pumpability in NGVAS carries a different flavor than in VASS. In VASS, the reachability algorithm checks for the existence of a cycle **up** which has a positive effect on (for simplicity) all counters, and similarly a cycle **dwn** which has a negative effect on all counters. The cycles may be different. Pumping in NGVAS instead asks for *one* derivation $S \to w_{\mathbf{up}}.S.w_{\mathbf{dwn}}$. To see how the one derivation leads to a different flavor, consider the NGVAS $S \to +1.S.-1 \mid \varepsilon$. Although it satisfies the pumping condition, the reachability set is bounded. This is different from pumping in VASS.

We distinguish two types of unboundedness and call a counter *vertically unbounded* if the pumpability condition holds, and *horizontally unbounded* if it is unbounded in the reachability set, like for normal VASS. To decide vertical unboundedness, we first reduce it to computing the downward closure of reachability sets, and hence horizontal unboundedness. The construction is similar to the Karp-Miller trees that are used to decide boundedness in VASS [25]. Afterwards, using Rackoff-like ideas [43], we reduce horizontal unboundedness to vertical unboundedness in NGVAS with fewer counters, which we can solve by induction.

Related Work. Already the emptiness problem for VASS turned out to be one of the hardest problems in Theoretical Computer Science and was studied for around 50 years. It was proven decidable in the 1980s [47, 41, 27, 28], but its complexity (Ackermann-complete) could only be determined very recently [38, 12, 13, 34]. Landmark results in this process were Leroux's new decision procedure based on inductive invariants [32, 33], and the insight that the classical KLMST decomposition, named after its inventors Sacerdote, Tenney, Mayr, Kosaraju, and Lambert [47, 41, 27, 28], computes run ideals [37].

Given this progress, problems that were considered out of reach moved into the focus of the community. One line of research asks whether reachability is decidable in models that generalize VASS, including VASS with nested zero tests and resets [44, 7, 8, 18, 39, 21], branching variants of VASS [52, 30, 14, 10], VASS with tokens that carry data [46, 24, 31, 6], valence systems [53], and amalgamation systems [1]. Another line asks whether we can compute information even more precise than reachability, like the downward closure of the reachability language [23], or separability by regular [26] or subregular languages [42, 9].

2 Preliminaries

2.1 Counters

Let I be a finite index set. Given a vector $v \in \mathbb{Z}^I$, we use $v[i]$ for the entry at dimension $i \in I$. We use $\|v\| = \sum_{i \in I} |v[i]|$ for the size. We write $0, 1 \in \mathbb{Z}^I$ for the vector with 0 resp. 1 in all dimensions. We write 1_i for the i-th unit vector that has 1 at dimension $i \in I$ and 0 otherwise. We also call vectors $v \in \mathbb{N}^I$ markings. We use $\leq$ to refer to the componentwise order on $\mathbb{N}^I$, where $v \leq w$ holds if $v[i] \leq w[i]$ for all $i \in I$.

The componentwise order on $\mathbb{N}^I$ forms a *well-quasi-order*, and our development occasionally relies on properties of well-quasi-orders. The most important is the following. For any infinite sequence $[v_i]_{i \in \mathbb{N}} \in (\mathbb{N}^I)^\omega$, there is an increasing subsequence $[v_{\phi(i)}]_{i \in \mathbb{N}}$, that is, $v_{\phi(i)} \leq v_{\phi(i+1)}$ holds for all $i \in \mathbb{N}$. We refer the reader to [17] for more information on the topic.

We augment the natural numbers with a top element $\omega = \sup \mathbb{N}$, and write $\mathbb{N}_\omega$ for $\mathbb{N} \uplus \{\omega\}$. We extend addition to $\mathbb{N}_\omega$ by $a + \omega = \omega + a = \omega$ for all $a \in \mathbb{N}_\omega$. A generalized marking is an element $v \in \mathbb{N}_\omega^d$. We use $\Omega(v) \subseteq [1, d]$ for the dimensions where v carries ω. A well-known concept from VAS reachability [28] that we will also need is the specialization quasi order on $\mathbb{N}_\omega$. It is defined by $\omega \sqsubseteq \omega$, $k \sqsubseteq k$, and $k \sqsubseteq \omega$ for all $k \in \mathbb{N}$, and lifted to generalized markings in a componentwise fashion.

2.2 Context-Free Grammars

A context-free grammar $G = (\Gamma, \Sigma, P, S)$ consists of a finite set of non-terminal symbols Γ, a finite set of terminal symbols Σ with $\Gamma \cap \Sigma = \emptyset$, a start non-terminal $S \in \Gamma$, and a finite set of productions $P \subseteq \Gamma \times (\Gamma \uplus \Sigma)^*$. We call sequences $\alpha, \beta \in (\Gamma \uplus \Sigma)^*$ of non-terminals and terminals sentential forms. We use $\alpha \xrightarrow{p} \beta$ for the derivation relation between sentential forms, which says that β can be obtained from α by an application of the production $p \in P$. We extend the relation to sequences of productions ps. We write $\alpha \xrightarrow{ps}$ if there is β so that $\alpha \xrightarrow{ps} \beta$ holds. We write $\alpha \to^* \beta$ if there is ps so that $\alpha \xrightarrow{ps} \beta$ holds. The grammar G defines the language $L(G) := \{\alpha \in \Sigma^* \mid S \to^* \alpha\}$.

Our development relies on the concept of strong connectedness. We say that $\Gamma_{sc} \subseteq \Gamma$ is *strongly-connected*, if for all $A, B \in \Gamma_{sc}$ there is a derivation $A \to^* \alpha.B.\beta$ for some $\alpha, \beta \in (\Gamma \uplus \Sigma)^*$. Note that each singleton $\{A\}$ is strongly connected under this definition. The grammar is strongly-connected, if Γ is strongly-connected.

The grammar is in weak Chomsky normal form (wCNF), if every production has at most two symbols on the right-hand side, $P \subseteq \Gamma \times (\Sigma \uplus \Gamma)^{\leq 2}$, and every terminal occurs on the right-hand side of a production. The advantage of this definition over the classical Chomsky normal form is that the notion of linearity applies without change. Namely, we say that the grammar is *linear*, if all productions have at most one non-terminal on the right-hand side. A non-terminal A is useful, if it can be used to derive a terminal word: there are $\alpha_1, \alpha_2 \in (\Gamma \uplus \Sigma)^*$ and $\alpha \in \Sigma^*$ so that $S \to^* \alpha_1.A.\alpha_2 \to^* \alpha$.

We need a linear-algebraic description of the derivations in a context-free grammar that can serve as an interface to VASS arguments. In particular, we need a way to capture the effect of derivations on the number of terminals and non-terminals. To this end, we define a variant of the Parikh image that is parameterized in the set it wishes to count. Let A be a set and $B, C \subseteq A$. We define $\psi_B : A^* \to \mathbb{N}^B$ as the function that maps a word $\alpha \in A^*$ to the vector $\psi_B(\alpha) \in \mathbb{N}^B$ which says how often each letter from B occurs in α. Consider productions $P \subseteq C \times A^*$. The effect of $p = (c, \alpha) \in P$ on the number of elements from B is $\Delta_{B,p} = \psi_B(\alpha) - \psi_B(c)$. The matrix $\Delta_B \in \mathbb{Z}^{B \times P}$ has $\Delta_{B,p}$ as column p.

We rely on a powerful theorem due to Esparza.

▶ **Theorem 1** (Theorem 3.1 in [15]). *Let G only have useful non-terminals, let $1 \leq v_P \in \mathbb{N}^P$. If $\Delta_\Gamma \cdot v_P \geq -1_S$, then there is $ps \in P^*$ with $S \xrightarrow{ps}$ and $\psi_P(ps) = v_P$.*

Theorem 1 allows us to turn a solution to a system of linear inequalities into a feasible production sequence. The following is the converse.

▶ **Lemma 2.** *Let $S \xrightarrow{ps} \alpha$ with $\psi_P(ps) = v_P$. Then $\psi_\Gamma(\alpha) = 1_S + \Delta_\Gamma \cdot v_P$ and $\psi_\Sigma(\alpha) = \Delta_\Sigma \cdot v_P$.*

For easy reference, we name some equations. By *Esparza-Euler-Kirchhoff* and its homogeneous variant, we mean

$$EEK(x_P): \ \Delta_\Gamma \cdot x_P \ = \ -1_S \qquad HEEK(x_P): \ \Delta_\Gamma \cdot x_P \ = \ 0 \ .$$

If $v_P \geq 1$ solves *EEK*, Theorem 1 yields a feasible production sequence. By Lemma 2, the resulting sentential form only consists of terminals. If v_P solves *HEEK*, the resulting sentential form has a copy of the start non-terminal. We also have equations that convert a number of productions into the number of terminals they produce. With $x_\Sigma \in \mathbb{N}^\Sigma$, we define

$$PT(x_P, x_\Sigma): \; x_\Sigma - \Delta_\Sigma \cdot x_P \; = \; 0 \; .$$

3 Proof Outline

We reduce PVASS reachability to emptiness for a new model called *nested grammar vector addition system*. For now, we can view an NGVAS N as a context-free grammar whose terminals are VAS updates, and then $R_{\mathbb{N}}(N)$ is the set of terminal words that lead from the initial to the final marking while staying non-negative. The reduction is similar to the translation of pushdowns to context-free grammars. Our main result is this.

▶ **Theorem 3.** *$R_{\mathbb{N}}(N) \neq \emptyset$ is decidable in $\mathfrak{F}_{\omega^{d+3}}$, where d is the number of counters.*

Here, $\mathfrak{F}_{\omega^{d+3}}$ is a class of problems that can be solved with multiply-recursive time and space. We refer the reader to [48] for an introduction to the field of fast-growing complexity.

Our decision procedure follows the KLMST approach to VASS reachability. We relax the non-negativity constraint in $R_{\mathbb{N}}(N) \neq \emptyset$ and admit runs that may fall below zero. We can then answer $R_{\mathbb{Z}}(N) \neq \emptyset$ with standard techniques from integer linear programming. Of course the relaxation is not sound in general. We identify a subclass of *perfect* NGVAS for which it is. It is convenient to incorporate the emptiness check into perfectness.

▶ **Proposition 4** (Iteration). *If N is perfect, $R_{\mathbb{N}}(N) \neq \emptyset$.*

The importance of perfectness stems from the fact that we can compute from every NGVAS a finite set of perfect NGVAS that reflects the set of non-negative runs. We call a set of NGVAS $\mathcal{D}$ a *deconstruction* of N, if it is finite and satisfies $R_{\mathbb{N}}(N) = R_{\mathbb{N}}(\mathcal{D})$. We also speak of a *perfect deconstruction*, if all $M \in \mathcal{D}$ are perfect. Our second main result is this.

▶ **Proposition 5** (Deconstruction). *There is a function $\mathsf{perf}\colon \mathsf{NGVAS} \to \mathbb{P}(\mathsf{NGVAS})$ so that $\mathsf{perf} \in \mathfrak{F}_{\omega^{d+3}}$ and, for all N, $\mathsf{perf}(N)$ is a perfect deconstruction of N.*

Theorem 3 now follows from Propositions 4 and 5:

$$R_{\mathbb{N}}(N) \neq \emptyset \qquad \text{iff} \qquad \mathsf{perf}(N) \neq \emptyset \; .$$

In the remainder of this overview, we explain how we define a function perf as required by Proposition 5. For this purpose, it is enough to view perfectness as the conjunction of five as of yet not relevant conditions we call *clean*, (R0), (R1), (R2), and (R3).

Similar to the KLMST decomposition for VASS reachability, perf is a recursive algorithm. It first checks whether the input N is already perfect. If so, it outputs $\{N\}$. Otherwise it computes a *decomposition* $\mathcal{D}$ of N, a deconstruction with $\mathsf{rk}(M) < \mathsf{rk}(N)$ for every $M \in \mathcal{D}$. Here, $\mathsf{rk}(N)$ is a well-founded rank. On the new NGVAS in $\mathcal{D}$, perf continues recursively. Since the rank is well-founded, the recursion terminates by König's Lemma.

To reduce the rank, perf has four subprocedures $\mathsf{dec}_{(\mathrm{R0})}, \mathsf{dec}_{(\mathrm{R1})}, \mathsf{dec}_{(\mathrm{R2})}, \mathsf{dec}_{(\mathrm{R3})} : \mathsf{NGVAS} \to \mathbb{P}(\mathsf{NGVAS})$, which check whether the conditions (R0), (R1), (R2), resp. (R3) hold and, if not, return a decomposition. Note that cleaning is not in this list. Cleaning can only guarantee that the rank does not increase and that all NGVAS in the output fulfill property *clean*. Being clean is a precondition for the other decompositions: all the procedures $\mathsf{dec}_{(\mathrm{R}i)} = \mathit{clean}.\mathsf{ref}_{(\mathrm{R}i)}$ first clean the input and then decrease the rank.

Before we can state the guarantees given by these functions, we have to highlight two differences to the case of VASS reachability, which have to do with the fact that we have to solve NGVAS emptiness in a highly recursive fashion. All our subprocedures rely on calling perf on inputs of smaller rank, which means all of them have the precondition that perf is reliable up to (and excluding) $\mathsf{rk}(N)$. Moreover, $\mathsf{ref}_{(\mathrm{R}i)}$ assumes that (R0) to (R$i-1$) hold.

▶ **Lemma 6.** *If perf is reliable up to* $\mathsf{rk}(N)$*, then* $\mathsf{clean}(N)$ *terminates with an output* $\mathcal{D}$ *that is a deconstruction of* N*, clean, and satisfies* $\mathsf{rk}(\mathcal{D}) \leq \mathsf{rk}(N)$*.*

▶ **Lemma 7.** *If perf is reliable up to* $\mathsf{rk}(N)$ *and the NGVAS* N *is clean and satisfies (R0) to (Ri − 1), then* $\mathsf{ref}_{(Ri)}(N)$ *decides whether condition (Ri) holds and, if not, outputs a decomposition.*

These lemmas imply termination of perf in Proposition 5 as sketched before. We elaborate on iteration, refinement, rank, and complexity.

4 NGVAS, Characteristic Equations, and Perfectness

We define the NGVAS model and perfectness. A *nested grammar vector addition system* $N = (G, Ct, Bo)$ is a context-free grammar $G = (\Gamma, \Sigma, P, S)$ in wCNF with a particular form of terminal symbols Σ and extra information Ct and Bo. Before we explain the individual components, we discuss the nesting structure of NGVAS. We implement the nesting via terminals, and define a notion of nesting depth. An NGVAS of nesting depth zero has as terminals counter updates in $\mathbb{Z}^d$. An NGVAS of nesting depth $n+1$ has as terminals NGVAS of depth at most n, also called childNGVAS. We call an NGVAS M a subNGVAS of N, if $M \in \Sigma$ or M is a subNGVAS of some $M' \in N.\Sigma$. An NGVAS (the underlying grammar) is expected to be strongly connected. We distinguish between NGVAS with linear grammars resp. non-linear grammars, and refer to the respective NGVAS as linear resp. non-linear. Our focus will be on the non-linear case.

The first piece of extra information $Ct = (Rs, Un)$ is the *(enforced) context* of the NGVAS. Enforced means the *semantics* forbids runs that do not respect the context. This will become clear in a moment. The enforced context consists of two components. The first is a linear set $Rs \subseteq \mathbb{N}^d \times \mathbb{N}^d$ of *source* and *target* markings, called the *restriction*, which overapproximates the run behavior. The restriction not only summarizes the effect of calls, as would be the case for procedure summaries. It actively enforces values for certain counters. We denote these values by $c_{in}, c_{out} \in \mathbb{N}_\omega^d$, which are defined as

$$c_{in}[i] \;=\; \sup\{v[i] \mid (v,w) \in Rs\} \qquad \text{and} \qquad c_{out}[i] \;=\; \sup\{w[i] \mid (v,w) \in Rs\}$$

for $i \in [1,d]$. Note that Rs is linear. This means the suprema are formed either over a singleton set or over an unbounded set.

The second component of the context is a set of counters $Un \subseteq [1,d]$ that are guaranteed to be unbounded anywhere in the NGVAS. To ensure this, we require the restriction to be closed as follows, $Rs + (1_i, 1_i) \subseteq Rs$ for all $i \in Un$. Moreover, children may only extend the Un component. Note that $Un \subseteq \Omega(c_{in}), \Omega(c_{out})$ follows from the definition.

The missing piece of extra information is the *boundedness information* $Bo = (D, in, out)$. We only present it for non-linear NGVAS, where it consists of a set D together with functions $in, out \colon \Gamma \to \omega^D \times \mathbb{N}^{[1,d]\setminus D}$. The set D contains counters that may be unbounded in the calling context. The functions in, out track, for every non-terminal A and bounded counter $i \notin D$, the values at the start resp. end of procedure A. If $A \to B.C$ is a production, then we

must have $in(A) = in(B), out(B) = in(C)$, and $out(C) = out(A)$. This is similar to tracking counters in the control-state of a VASS [28]. The context information of a child has to comply with boundedness information of its parent: the set D of the parent becomes the set Un of the child, and if a non-terminal $A \in \Gamma$ produces a terminal $\sigma \in \Sigma$ with a rule $A \to \alpha\sigma\beta$, then $\sigma.c_{in} = in(A)$ if $\alpha = \varepsilon$ and $\sigma.c_{out} = out(A)$ if $\beta = \varepsilon$.

The NGVAS inherits the notion of a language from the underlying grammar, $L(N) = L(G)$. We also associate with the NGVAS a set of runs $R_{\mathbb{N}}(N) \subseteq \mathbb{N}^d \times (\mathbb{Z}^d)^* \times \mathbb{N}^d$. The start- and final-markings of these runs should satisfy the given restriction. As a consequence, the run should be enabled in $N.c_{in}$ and lead to $N.c_{out}$. These requirements are made not only for N, but for all subNGVAS. The definition is by Noetherian induction:

$$R_{\mathbb{N}}(N) = \{(v, r, w) \in R_{\mathbb{N}}(\alpha) \mid \alpha \in L(G) \wedge (v, w) \in Rs\} .$$

Here, $R_{\mathbb{N}}(\alpha)$ is defined by concatenating the runs of $\alpha[1]$ to $\alpha[|\alpha|]$, provided they agree on the intermediary markings. That is, $R_{\mathbb{N}}(\varepsilon) = \{(v, \varepsilon, v) \mid v \in \mathbb{N}^d\}$ and

$$R_{\mathbb{N}}(\alpha_0.\alpha_1) = \{(v_0, r_0.r_1, w_1) \mid (v_0, r_0, w_0) \in R_{\mathbb{N}}(\alpha_0) \text{ and } (w_0, r_1, w_1) \in R_{\mathbb{N}}(\alpha_1)\} .$$

If a terminal is an update, we use $R_{\mathbb{N}}(u) = \{(v, u, v + u) \mid v, v + u \in \mathbb{N}^d\}$. If a terminal is a childNGVAS, it has a lower nesting depth and so the set of runs is already defined by the induction hypothesis. Note that the definition yields a strong form of mononoticity on the counters in Un. We have $(v + y, r, w + y) \in R_{\mathbb{N}}(N)$, if $(v, r, w) \in R_{\mathbb{N}}(N)$ and $y \in \mathbb{N}^d$ with $y[i] = 0$ for all $i \notin Un$. This holds as no subNGVAS is allowed to constrain the value of a counter $i \in Un$. This monotonicity allows us to treat counters that are unbounded in a parent as integer counters in a child: a run in the child will be enabled provided the parent gives large enough values to the counters in Un. To fix the notation, for a non-terminal A, we also define $R_{\mathbb{N}}(A)$ as the union of $R_{\mathbb{N}}(\alpha)$ over all α that can be derived from A.

Finally, we define the set of $\mathbb{Z}$-runs $R_{\mathbb{Z}}(N) \subseteq \mathbb{Z}^d \times (\mathbb{Z}^d)^* \times \mathbb{Z}^d$ as

$$R_{\mathbb{Z}}(N) = \{(v, r, w) \in R_{\mathbb{Z}}(\alpha) \mid \alpha \in L(G) \wedge \exists y \in \mathbb{Z}^d.\ (v + y, w + y) \in Rs\} .$$

Here, $R_{\mathbb{Z}}(\alpha_0.\alpha_1)$ and $R_{\mathbb{Z}}(u)$ are defined similar to the $\mathbb{N}$-case but with a $\mathbb{Z}$ relaxation. Importantly, for runs in $R_{\mathbb{Z}}(\alpha_0.\alpha_1)$ the connecting marking may become negative.

4.1 The Characteristic Equation

The decision procedure for VASS reachability approximates the VASS at hand by a system of linear equations, called the characteristic system [47, 41, 27, 28, 38]. The characteristic system should be imagined as a variant of the marking equation. The reachability algorithm refines the VASS until the characteristic equation closely reflects its behavior. This reduces the VASS reachability problem to the problem of solving a system of linear equations. Since we argue along the same lines, we also need a characteristic system for NGVAS.

Consider the non-linear NGVAS $N = (G, Rs, Un, Bo)$. Let $U \subseteq \mathbb{Z}^d$ be the set of updates that appear anywhere in N. We approximate the set of runs that solve reachability with the *characteristic system of equations CHAR* defined as

$$EEK(x_P) \wedge REACH(x_P, x_U, x_{in}, x_{out}) \wedge Rs(x_{in}, x_{out}, x_V).$$

The equations $REACH$ evaluate reachability as follows:

$$PT(x_P, x_\Sigma) \wedge UPD(x_\Sigma, x_U) \wedge ME(x_{in}, x_U, x_{out}).$$

We have variables $x_P \in \mathbb{N}^P$ that determine how often each production should be taken. This choice has to satisfy Esparza-Euler-Kirchhoff. The productions lead to a number of terminal symbols given by $x_\Sigma \in \mathbb{N}^\Sigma$. We introduce variables $x_U \in \mathbb{N}^U$ that store how often each update should be used, in N and in the subNGVAS. The constraint $UPD(x_\Sigma, x_U)$ fills x_U with appropriate values, and we elaborate on it in a moment. We have variables x_{in} and x_{out} that store the input and output markings for which the approximation of reachability holds. We make sure they are related via x_U by the marking equation. The choice of updates also has to satisfy the restriction. Let x_V state how often each period in that linear set will be used. The constraint $Rs(x_{in}, x_{out}, x_V)$ now checks whether applying the base vector and the periods as dictated by x_V yields (x_{in}, x_{out}). This also ensures $x_{in} \sqsubseteq c_{in}$ and $x_{out} \sqsubseteq c_{out}$, since periods may not modify the non-ω counters in c_{in} or c_{out}.

The constraint $UPD(x_\Sigma, x_U)$ has to meet the following specification: the variables x_U store how often each counter update from U is used in a run that can be derived from the number of terminal symbols x_Σ. The challenge is to include the updates generated by subNGVAS. We achieve this by considering the restriction of the immediate childNGVAS. For each instance of a childNGVAS, we must have a repetition of the base vector and an arbitrary number of period vectors. We realize this constraint by a set of linear equations that relate x_Σ to x_U.

We also need a *homogeneous variant* $HCHAR$ of the characteristic equations,

$$HEEK(x_P) \ \wedge \ REACH(x_P, x_U, x_{in}, x_{out}) \ \wedge \ HRs(x_{in}, x_{out}, x_V) \ .$$

We check reachability between the zero versions of the input and output markings, and we use the homogeneous variants of Esparza-Euler-Kirchhoff and the restriction. The homogeneous characteristic equations are defined such that if s solves $CHAR$ and h solves $HCHAR$, then also $s + h$ solves $CHAR$. To be explicit, we consider $\mathbb{N}$ solutions. Using well-quasi-ordering arguments, one can then show that the variables which are unbounded in the solution space of $CHAR$ are precisely the variables that receive a positive value in some homogeneous solution. This leads to the definition of the *support* of the characteristic equations

$$supp(HCHAR) \ = \ \{x \in vars(HCHAR) \ \mid \ \exists h.\ h \text{ solves } HCHAR \wedge h(x) > 0\} \ .$$

As the solution space of $HCHAR$ is closed under addition, there always is a *full homogeneous solution* that gives a positive value to all variables in the support.

4.2 Perfectness

We now define the refinement conditions (R0), (R1) and (R3) that will be established by perf. We also define cleanness conditions (CX) that will be established by clean. We call an NGVAS that satisfies all these conditions *perfect*. We focus on the interesting case of non-linear NGVAS, and omit (R2) which only concerns the linear case. We also omit minor cleanness conditions.

(C0) For every $(v_{in}, v_{out}) \in Rs$ there is a solution s to $CHAR$ with $s[x_{in}] = v_{in}$, $s[x_{out}] = v_{out}$. For every $w_V \in \mathbb{N}^V$ with $w_V \geq 1$ there is a full homogeneous solution h to $HCHAR$ with $(h[x_{in}], h[x_{out}]) = V \cdot w_V$.

(C3) The base effects of all (in the linear case, recurring) childNGVAS are enabled, for all $M \in \Sigma$ with restriction $(v_{M,in}, v_{M,out}) + V_M^*$ there is $(v_{M,in}, r, v_{M,out}) \in R_\mathbb{N}(M)$.

(R0) All subNGVAS $M \in N$ are perfect.

(R1) All productions and all period vectors in the restrictions of (in the linear case, recurring) childNGVAS $M \in \Sigma$ are in support. That is, $x_P, x_{M,V} \subseteq supp(HCHAR)$.

(R3) If a counter should be unbounded, $i \in D$, then it actually is unbounded in both directions: there exists a derivation $S \to^* r_{up}.S.r_{dn}$ so that r_{up} can be fired at c_{in} and increases i, and r_{dn} can be fired backwards at c_{out} and firing backwards increases i. The latter has the purpose of pumping down the counter.

5 Iteration Lemma: Proof of Proposition 4

To prove the iteration result, we use an induction on the nesting depth. We slightly strengthen the inductive statement as follows:

▶ **Theorem 8.** *Consider a perfect NGVAS N with $Rs = (v_{in}, v_{out}) + V^*$. Consider $w \in Rs$ and $w_V \in \mathbb{N}^V$ with $w_V \geq 1$. There is $k_0 \geq 1$ so that for every $k \geq k_0$ there is a run $(v_{in}^{(k)}, r^{(k)}, v_{out}^{(k)}) \in R_{\mathbb{N}}(N)$ with the source-target pair $(v_{in}^{(k)}, v_{out}^{(k)}) = w + k \cdot V \cdot w_V$.*

This says that every source-target pair in the restriction, $w \in Rs$, can be implemented by an $\mathbb{N}$-run as long as we add a given loop w_V often enough. The loop w_V may even be chosen arbitrarily, except that it has to use every period of Rs. We now give a proof sketch of the induction step. We assume that the NGVAS N is non-linear, but remark that many aspects carry over to the linear case. For the sake of simplicity, we also assume $\Omega(N.c_{in}) = \Omega(N.c_{out}) = \emptyset$. Some ideas are based on KLMST, though with extra difficulties.

By cleanness condition (C0) and standard VASS arguments, there is a $\mathbb{Z}$-run $r_{\mathbb{Z}}$ with the desired effect w. Our goal is to turn $r_{\mathbb{Z}}$ into an $\mathbb{N}$-run. We discuss some of the problems, solutions, and the shape of the result.

▶ **Problem 1.** *Counters may go negative on* the current level *of nesting.*

To solve this problem, we adapt the run to $r_{up}^k.r_{\mathbb{Z}}.r_{dn}^k$, where r_{up}, r_{dn} are the loops from (R3). This pumps all counters to high values so that $r_{\mathbb{Z}}$ does not go negative if k is large enough.

▶ **Problem 2.** *Counters may go negative on a* lower level of nesting.

The problematic situation is that a counter i is bounded on the current but unbounded on a lower nesting level. Then the solution to Problem 1 does not apply, but we have to pump the counter *inside the child*. To do so, we rely on (R0). Since all childNGVAS M are perfect, we can invoke the induction hypothesis. It modifies $r_{\mathbb{Z}}$ to a run $r_{\mathbb{Z}}^{(k)}$ by adding periods of $M.Rs$ that make sure the run is enabled on the lower level. At this point, the shape is $r_{up}^k.r_{\mathbb{Z}}^{(k)}.r_{dn}^k$. There are two problems left.

▶ **Problem 3.** *Both additions change the effect of $r_{\mathbb{Z}}$.*

We may have $\mathit{eff}(r_{up}) + \mathit{eff}(r_{dn}) \neq 0$, and also adding the periods to obtain $r_{\mathbb{Z}}^{(k)}$ may change the effect. To still reach the target c_{out}, we add a loop $S \to^* r_{dif1}.S.r_{dif2}$ to the run which cancels out the changes. The construction is as follows. Similar to KLMST and using (R1), we can determine a loop which, together with the aforementioned modifications, sums up to a homogeneous solution. The construction is trickier than it may seem. The characteristic system only tracks the number of instances of childNGVAS, but cannot track the assignment of periods to children. So the loop may have to construct runs for children that do not receive any periods. For these children, we rely on (C3): already the base effect of the child can be realized by a valid run. This idea is due to [44] and also occurs in [22]. At this point, our run is $r_{up}^k.r_{dif1}^k.r_{\mathbb{Z}}^{(k)}.r_{dif2}^k.r_{dn}^k$.

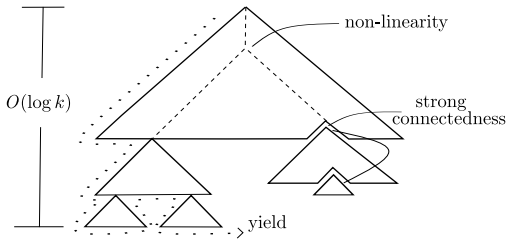
We focus on a new problem.

▶ **Problem 4.** *r_{dif1} may have a negative effect which r_{up} cannot make up for.*

The problem does not have an analogy in the VASS setting. To solve it, we modify the run so that it can still be derived in the grammar but also stays non-negative:

$$r^{(k)} \quad = \quad r^k_{up}.\pi^{(k)}(r^k_{dif1}.r^{(k)}_{\mathbb{Z}}.r^k_{dif2}).r^k_{dn} \ .$$

Here, $\pi^{(k)}$ is a permutation of the inner sequence that we obtain with a new result for context-free grammars (no VASS involved) called the *wide tree theorem*. To explain it, consider the derivation $S \to r_{dif1}.S.r_{dif2}$ that we want to repeat k-times. The theorem says that we can find a parse tree for the repetition whose height only grows logarithmic in k. Behind this is a binary tree construction, illustrated below, which fundamentally needs non-linearity and strong connectedness.



To formalize the wide tree theorem, consider a parse tree t with $k \geq 1$ copies of a production vector $v_P \in \mathbb{N}^P$, $\psi_P(t) = k \cdot v_P$. We introduce functions that track the provenance of Σ-labeled leaves in t. Formally, $prov : yield(t) \to [1, k]$ *tracks provenance*, if $\psi_\Sigma(prov^{-1}(i)) = \Delta_\Sigma \cdot v_P$ for all $i \in [1, k]$. The image $prov(\ell)$ of leaf $\ell \in yield(t)$ is meant to tell us which copy of v_P produced the leaf, $prov^{-1}(i) \subseteq yield(t)$ is the subword of all leaves produced by copy i of v_P, and $\psi_\Sigma(prov^{-1}(i))$ is the Parikh image wrt. Σ. Consider a prefix α of $yield(t)$. We say that copy i of v_P is *complete* in α, if α contains all leaves produced by copy i, $prov^{-1}(i) \subseteq \alpha$. Otherwise, if also the remainder of the yield contains terminals produced by this copy, we say that copy i is *incomplete* in α. The *order* of the provenance tracking function is the maximal number of incomplete copies of v_P in any prefix of the yield. It is worth noting that the provenance tracking function is not actually tied to the shape of t, it just gives information about the terminal symbols that will be encountered while processing the yield. This suffices for our purposes. However, in the proof of the wide tree theorem, we construct a provenance function that indeed follows the structure of the constructed parse tree.

▶ **Theorem 9.** *Consider a context-free grammar G that is non-linear, strongly connected, and only has useful non-terminals. Let $v_P \geq 1$ solve HEEK(x_P). For every $k \geq 1$ there is a parse tree t_k with $\psi_P(t_k) = k \cdot v_P$ and height $h(t_k) \leq \lceil 1 + \log_2 k \rceil \cdot \|v_P\|$. Moreover, t_k admits a provenance tracking function of order at most $\lceil 1 + \log_2 k \rceil$.*

The relevance to our problem is this. As $r_{dif1} + r_{dif2}$ was chosen to compensate $r_{up} + r_{dn}$, we have the equality $\mathit{eff}(r_{up} + r_{dif1} + r_{dif2}) = -\mathit{eff}(r_{dn}) > 0$. Hence, whenever we finish executing a pair (r_{dif1}, r_{dif2}), we know this caused a positive effect. Only copies of r_{dif1} that are incomplete in that we have not yet executed r_{dif2} may lead to negative counters. In the parse tree constructed by the wide tree theorem, only *logarithmically* in k many copies of r_{dif1} can be incomplete. Since r^k_{up} produces a linear in k positive value on every counter, and linear beats logarithmic, $r^{(k)}$ is an $\mathbb{N}$-run for k large enough.

6 Decomposition

It remains to sketch the proofs of Lemmas 6 and 7. Lemma 6 states that, given an NGVAS which is not clean, we can clean it without increasing the rank. Lemma 7 states that, given a clean NGVAS that is not perfect, we can compute a decomposition. Before we can elaborate on the proofs, we have to discuss the rank that underlies the decomposition.

To make use of the rank, we need properties that go beyond well-foundedness. We need childNGVAS to have a lower rank, and more unconstrained counters to also yield a lower rank. The intuition is that unconstrained counters are easy (in VASS terms, they are $\mathbb{Z}$-counters).

▶ **Lemma 10.** *Let M be a childNGVAS of a* non-linear *NGVAS N. Then $\mathsf{rk}(M) < \mathsf{rk}(N)$.*
Let M be a childNGVAS of an NGVAS N. Then $\mathsf{rk}(M) \leq \mathsf{rk}(N)$.
Let N_1, N_2 be two NGVAS with $|N_1.Un| > |N_2.Un|$. Then $\mathsf{rk}(N_1) < \mathsf{rk}(N_2)$.

To guarantee these properties, we rely on the so-called *cycle space* of an NGVAS, similar to the rank for VASS defined by Leroux and Schmitz in [38]. As Lemma 10 suffices to understand the algorithm, we postpone the definition of the rank to Section 8.2.

With Lemma 10 at hand, we are ready to inspect the cleanness and perfectness conditions one-by-one, and explain how to decrease rank in each case. The first property (C0) is similar to *saturatedness* in [38]: the restrictions Rs play the role of the input and output markings of an SCC in a VASS, though now it is an input-output relation. The action we take remains the same, which we illustrate with the following example. Say that a period is bounded, and the characteristic system has solutions that repeat this period 2, 3, and 5 times. Then we remove the period and create three copies of the NGVAS, where the base of the linear set Rs contains 2, 3, and 5 copies of this period, respectively.

We move on to condition (C3). With the help of the omitted cleanness conditions, we can assume that the childNGVAS M at hand is perfect, but it does not have a run for the input-output pair that forms the basis of the restriction, and therefore the parent N is not perfect. To establish (C3), we first find the minimal elements of the reachability relation $R = \{(v_{in}, v_{out}) \mid (v_{in}, r, v_{out}) \in R_{\mathbb{N}}(M)\}$ of the child M. Then, to preserve the set of runs of M, we create one copy of the child for each $(v_{in}, v_{out}) \in \min R$, where we modify the base pair of M to (v_{in}, v_{out}). We compute $\min R$ in an iterative fashion. Let X be the current under-approximation. We have $X = \min R$, if the complement of the upward closure of X has an empty intersection with the reachability relation of M. This can be checked by forming an NGVAS M' and calling perf on it. The call returns the empty set (of NGVAS) if and only if there is no reaching run left. Note that the direction from right to left needs Theorem 8. Even more, if there is a reaching run, Theorem 8 gives us a short one, which we can find by enumeration. The run adds an element to X and we repeat.

We proceed with the refinement conditions. Condition (R0) requires all children M to be perfect, and the solution is to apply perf on M. In case of non-linear NGVAS, Lemma 10 guarantees that $\mathsf{rk}(M) < \mathsf{rk}(N)$, justifying the recursive call. In case of linear NGVAS the argument is slightly more involved: the rank remains the same, but the nesting depth decreases, again justifying the recursion. We defer these details of the rank to Section 8.2.

Condition (R1) is again similar to VASS: if a production or period is bounded, say by $B \in \mathbb{N}$, then we create $B + 1$ copies of the current NGVAS with this production or period being redirected to the next copy. As for VASS, if *all* bounded productions and periods are removed simultaneously, then the cycle space and hence the rank decreases.

Finally, we arrive at (R3) which is provably difficult to even *decide*, and we devote all of Section 7 to this decomposition.

7 Up- and Down-Pumping

In this section, we explain how to decide (R3) and compute a decomposition if it does not hold. Before doing so, we briefly discuss why implementing $\mathsf{ref}_{(R3)}$ is difficult. The difficulty stems from the similarity between the reachability and the pumpability questions. This is illustrated by the following sequence of reductions. The reachability problem with d counters reduces to the question whether there is a run from 0 to 0, which reduces to the question whether an NGVAS with $d+1$ counters has any run starting at 0, which reduces to boundedness. The first and the last reduction are straightforward, the second was first observed in [29] and is as follows. The extra dimension is used as a budget. It is initialized with the production $S' \to (1_{d+1}).S'.(-1_{d+1})$ setting the counter to an arbitrary value which, however, has to be returned at the end. Throughout the computation, the budget is updated to reflect $-\sum_{i\in[1,d]} v[i]$ where v is the effect on the other counters. Since the initial marking is 0, this implies that $\sum_{i\in[1,d]} v_{curr}[i] + b_{curr} = b_{init}$ always holds, where v_{curr} is the current effect, b_{curr} the current budget, and b_{init} the initial budget. Since we have to return the initial budget at the end, the final marking must also be 0. The discussion gives an important hint for implementing $\mathsf{ref}_{(R3)}$: we will have to use recursive reachability checks.

Recall that we may assume the input N fulfills all perfectness conditions except $(R3)$ and that perf is reliable up to the rank of the input. The function has to check whether there are up- and down-pumping runs as required by $(R3)$, and if not it has to compute a decomposition $\mathcal{D}$ that is of smaller rank than the input.

Our insight is that both the check for pumping runs and the decomposition can be reduced to computing two functions. Let $\mathsf{C} = \{v \in \mathbb{N}_\omega^d \mid Un \subseteq \Omega(v) \subseteq D\}$ contain the generalized markings of interest and $\mathsf{Dom} = \mathsf{C} \times (\Gamma \cup \Sigma)$. We let $\mathsf{reach}, \mathsf{breach} : \mathsf{Dom} \to \mathbb{P}(\mathbb{N}_\omega^d)$ with

$$\mathsf{reach}(v,\sigma) = \mathsf{Id}(\downarrow\{z \in \mathbb{N}^d \mid (y,r,z) \in R_{\mathbb{N}}(\sigma),\ y \sqsubseteq v\})$$
$$\mathsf{breach}(w,\sigma) = \mathsf{Id}(\downarrow\{y \in \mathbb{N}^d \mid (y,r,z) \in R_{\mathbb{N}}(\sigma),\ z \sqsubseteq w\}).$$

Function reach takes as input a generalized marking v and a terminal or non-terminal symbol σ. It considers all concrete markings y represented by v and computes the set of reachable markings, more precisely, the set of all markings reachable via runs that can be derived from σ. To obtain a finite representation, the function closes the set downwards ($\downarrow-$) and computes the decomposition into maximal ideals (Id). The downward closure does not lose information when it comes to pumping. The maximal ideals in a well-quasi-order are guaranteed to form a finite set, and in our case correspond to generalized markings [17].

We give the reduction and then the computability of breach and reach. The latter is a main achievement.

7.1 Computing $\mathsf{ref}_{(R3)}$

The function first has to check the existence of an up-pumping run r_{up} from c_{in} and a down-pumping run r_{dn} from c_{out} so that $S \to^* r_{up}.S.r_{dn}$ holds. Assuming the computability of breach and reach, we can use a mild generalization of the Karp-Miller tree.

The Karp-Miller tree is labeled by triples (v_1, A, v_2) from $\mathbb{N}_\omega^d \times \Gamma \times \mathbb{N}_\omega^d$. The non-terminal A should be understood as to lie on the path $S \to^* r_{up}.S.r_{dn}$. The markings v_1 and v_2 represent the current outcome of r_{up} and r_{dn}, computed using reach on c_{in} resp. breach on c_{out}. To be precise, the root is (c_{in}, S, c_{out}). We extend a node (v_1, A, v_2) by considering all rules $A \to B_1.B_2$. This can be understood as selecting the parse tree that contains the pumping situation. We consider both choices $B = B_1$ and $B = B_2$, which corresponds to selecting the

path in this parse tree. Let $B = B_2$. We consider all $w_1 \in \mathsf{reach}(v_1, B_1)$. With these choices made, we define the successor node in the tree as (w_1, B, v_2). We achieve termination with an acceleration step. We look for a node (w_1', B, v_2') on the path to the new successor so that $(w_1', v_2') < (w_1, v_2)$ holds. Then we replace the entries in our successor by ω wherever the inequality is strict.

▶ **Lemma 11.** *Assume* reach *and* breach *are computable for N. Then the Karp-Miller tree construction terminates. Moreover, N satisfies (R3) if and only if the tree contains a node (v_1, A, v_2) s.t. $v_j[i] = \omega$ for all $i \in D$ and $j \in \{1, 2\}$.*

If there is no pumping run, we have to decompose the NGVAS. Unfortunately, the Karp-Miller tree is not precise enough for this: it mixes parse trees, has different views on the same parse tree, and only has incomplete information about runs.

We compute the decomposition from a new type of context-free grammar called a *coverability grammar*. The idea is to restrict the NGVAS by reach and breach. We implement this restriction with two triple constructions that are inspired by the intersection of a context-free grammar with a regular language. The non-terminals in the coverability grammar are 5-tuples (v_1, v_2, A, w_2, w_1) from $\mathbb{N}_\omega^d \times \mathbb{N}_\omega^d \times \Gamma \times \mathbb{N}_\omega^d \times \mathbb{N}_\omega^d$. When denoted as the more familiar triple, (v_1, A, v_2) says that A can produce a run which transforms v_1 into v_2, meaning $v_2 \in \mathsf{reach}(v_1, A)$. The triple (w_2, A, w_1) should be read similarly but with $w_2 \in \mathsf{breach}(w_1, A)$. For a production $A \to B_1.B_2$ in the NGVAS, the coverability grammar will have

$$(v_1, v_2, A, w_2, w_1) \;\to\; (v_1, v_2', B_1, w_3, w_2').(v_2', v_3, B_2, w_2', w_1) \;.$$

The markings are chosen as expected, $v_2' \in \mathsf{reach}(v_1, B_1)$ is the start marking for the first triple in the second non-terminal, and we again compute $v_3 \in \mathsf{reach}(v_2', B_2)$. For the other triple, the reasoning is similar. The new source and target markings have to respect the reachability that was promised initially, meaning we only add the production if $v_3 \sqsubseteq v_2$ and $w_3 \sqsubseteq w_2$ hold. Note that we cannot require equality here, as the specific production $A \to B_1.B_2$ may cause additional counters to become bounded compared to our original estimations v_2 resp. w_2. There is again an acceleration step that guarantees finiteness. Note that we do not restrict v_3 to w_1 and w_3 to v_1. With such a restriction, the second triple would eliminate ω entries from the first and vice versa. Then the number of ω entries would not grow monotonically, and we would lose termination. As we have defined it, the two triple constructions do not influence each other.

The *decomposition* consists of a single NGVAS. We split the coverability grammar into its strongly-connected components, and use them to create the nesting structure.

We have to determine the boundedness information for the new NGVAS. Recall that generalized markings represent downward-closed sets. We can implement the intersection of these sets by an elementwise minimum wrt. $\sqsubseteq$ on the generalized markings. As boundedness information for (v_1, v_2, A, w_2, w_1), we use $v_1 \cap w_2$ and $v_2 \cap w_1$. This is the most precise information we can obtain from the two triples.

To see that the rank decreases, we prove that in every SCC some counter i is bounded (as opposed to VASS, however, there may be no counter i which is bounded in every SCC). To see this, observe that the corresponding node in the Karp-Miller graph would be of the form (v_1, A, v_2) with $v_j[i] = \omega$ for all $i \in D$ and $j \in \{1, 2\}$, and hence (R3) would hold. Like for VASS [38], we will use the cycle space of SCCs as the rank. Bounding a counter decreases the cycle space of the corresponding SCC, and decreasing all cycle spaces decreases the rank.

Before we state the formal guarantees given by the decomposition, we generalize the definition of the coverability grammar.

7.2 Generalization

Rather than defining the coverability grammar only for the functions breach and reach, we take two such functions as parameters and define $CG(N, \mathsf{apre}, \mathsf{apost})$ relative to them. We expect that apre and apost are *sound approximations* of breach resp. reach for N, meaning they are computable and over-approximate the original functions (in a precise sense). We use the notation N_G for the decomposition computed from $G = CG(N, \mathsf{apre}, \mathsf{apost})$.

▶ **Lemma 12.** *Let* apre *and* apost *be sound approximations of* breach *resp.* reach *for* N. *Then the computation of* N_G *terminates. Moreover, if* N *does not satisfy* $(R3)$ *even in this approximation, then* N_G *is a decomposition of* N.

7.3 Computing (b)reach: Simple Cases

We now prove the computability of breach and reach that is behind our implementation of $\mathsf{ref}_{(R3)}$. We focus on reach as the reasoning for breach is similar. Our strategy is again to establish assumptions as strong as possible. For the input NGVAS, we already have all perfectness conditions except $(R3)$ and the reliability of perf up to its rank. We now introduce assumptions (a) to (d) and show that every single one of them makes reach easy to compute. What we gain by this is that we can assume $\neg(\mathsf{a}) \wedge \neg(\mathsf{b}) \wedge \neg(\mathsf{c}) \wedge \neg(\mathsf{d})$ when proving the computability for the remaining hard cases. Fix a marking-symbol pair $(v, \sigma) \in \mathsf{Dom} \subseteq \mathsf{C} \times (\Sigma \cup \Gamma)$. We show how to compute $\mathsf{reach}(v, \sigma)$ under these assumptions:

(a) The query refers to a childNGVAS $\sigma \in \Sigma$.
(b) The source marking has an additional ω, $Un \subsetneq \Omega(v) \subseteq D$.
(c) Condition (R3) already fails in an approximation that ignores one counter on the input side as well as the output side.
(d) Condition (R3) already fails in a $\mathbb{Z}$-approximation.

With assumption (a), we have $\sigma \in \Sigma$. ChildNGVAS have a lower rank, so the reliability of perf allows us to compute a perfect decomposition. From this decomposition, we can read-off the values of reach.

Assumption (b) gives us $Un \subsetneq \Omega(v) \subseteq D$. We construct an NGVAS $[v, \sigma, out(\sigma)]_N$ that still has $\mathsf{reach}(v, \sigma)$ as the reachable markings but a smaller rank. On this NGVAS, we reason as in (a). The construction simply changes the input marking to v, the initial non-terminal to σ, and replaces the set of unconstrained counters by $\Omega(v)$. The latter is what makes the rank go down.

From now on, we can assume $Un = \Omega(v) \subseteq D$, the negation of (b). We call counters from $D \setminus Un$ *relevant* as they should be pumped. Assumption (c) is that (R3) even fails in an approximation that ignores relevant counters. More precisely, we have a family of approximations, one for each pair of functions apost_i, apre_j that ignore the relevant counters i resp. j. Ignoring these counters means we set them to ω in the given marking. The computability of apost_i, apre_j follows as in (b). Given their computability, we can check the failure of (R3) using the Karp-Miller tree. Lemma 12 now yields a decomposition that reduces the rank. We then proceed with a perfect decomposition as in (a).

Finally, let (d) hold. That is, (R3) fails in an approximation that does not have to keep the counters non-negative. The underlying functions $\mathsf{apre}_{\mathbb{Z}}$ and $\mathsf{apost}_{\mathbb{Z}}$ are trivially computable. To check (R3), we do not need the Karp-Miller tree but can use integer linear programming. We then reason as in (c).

The approximations in (c) and (d) are incomparable. The approximation based on $\mathsf{apost}_i, \mathsf{apre}_j$ looks for runs that pump all relevant counters except i and j. These pumping runs are guaranteed to remain non-negative on all counters except i resp. j. On i resp. j,

they are allowed to fall below zero and even have a negative total effect. The approximation based on $\mathsf{apre}_{\mathbb{Z}}, \mathsf{apost}_{\mathbb{Z}}$ looks for runs that pump all relevant counters. These pumping runs are allowed to fall below zero on any counter, but guarantee a non-negative effect.

7.4 Computing (b)reach: Hard Case 1

We show how to compute $\mathsf{reach}(v, A)$ with $\Omega(v) = Un$ for an NGVAS that is almost perfect: we even have up- and down-pumping runs as soon as we ignore an arbitrary pair of relevant counters (since (c) failed), and we have up- and down-pumping runs for all relevant counters if we admit integer values (since (d) failed).

We proceed with a Rackoff-like case distinction [43]. We show that we can compute a bound Bd that satisfies the following. If in the input marking v the value of *some* relevant counter exceeds Bd, then we show that we can find pumping runs by stitching together the almost pumping runs. This means the NGVAS is perfect and we can read-off $\mathsf{reach}(v, A)$ as in (a). If in marking v all relevant counters are bounded by Bd, then we show how to compute $\mathsf{reach}(v, A)$ in the next section.

We explain how the almost pumping runs lead to a full pumping run provided we have a high value, say in the relevant counter i. Consider marking $v = c_{in}$ and non-terminal $A = S$. Let r^i_{up} be the up-pumping run that ignores counter i and let $r^{\mathbb{Z}}_{up}$ be the up-pumping run that may fall below zero. For the counters that are tracked concretely in the NGVAS, r^i_{up} and $r^{\mathbb{Z}}_{up}$ are enabled and have effect zero. We can ignore these counters in our analysis. The same holds for the counters in $\Omega(c_{in})$. For simplicity, let the counter updates stem from $\{-1, 0, 1\}$. With this assumption, $r^{\mathbb{Z}}_{up}$ is enabled as soon as we have a value of $|r^{\mathbb{Z}}_{up}|$ in every relevant counter. Moreover, $|r^i_{up}|$ is a bound on the negative effect that r^i_{up} may have on counter i. Then

$$r_{up} \quad = \quad (r^i_{up})^{|r^{\mathbb{Z}}_{up}|}.(r^{\mathbb{Z}}_{up})^{|r^{\mathbb{Z}}_{up}|\cdot|r^i_{up}|+1}$$

has a positive effect on all relevant counters. Indeed, the only negative effect is due to r^i_{up}, and this is compensated by the repetition of $r^{\mathbb{Z}}_{up}$. The run is enabled from c_{in} as soon as we have value $|r^{\mathbb{Z}}_{up}| \cdot |r^i_{up}| + |r^{\mathbb{Z}}_{up}|$ in counter i. Remember that we have to derive the up- and down-pumping runs together. Let k be the maximum of $|r^{\mathbb{Z}}_{up}|$ and $|r^{\mathbb{Z}}_{dn}|$, and let $k_{i,j}$ be the maximum of $|r^i_{up}|$ and $|r^j_{dn}|$.

▶ **Lemma 13.** *Let N be perfect except for (R3) and assume $\neg$(b) $\wedge \neg$(c) $\wedge \neg$(d) holds. If there are relevant counters i and j so that $c_{in}[i], c_{out}[j] \geq k \cdot (k_{i,j} + 1)$, then N is perfect.*

The lemma refers to a single NGVAS. To compute the function reach, we need a lower bound on the values of (arbitrary pairs) i and j that works for all NGVAS $[v, A, out(A)]_N$, where the given v is the initial marking and the given A is the initial non-terminal. As there are only finitely many non-terminals, we can consider a fixed A. The runs $r^{\mathbb{Z}}_{up}$ and $r^{\mathbb{Z}}_{dn}$ do not have to stay non-negative. This means they only depend on the set of relevant counters, but not on the choice of v. As the set of relevant counters is also fixed, we can consider an arbitrary pair of up- and down-pumping $\mathbb{Z}$-runs. Let k be a bound on their length.

The challenge is to compute a bound on the length of r^i_{up} and r^j_{dn}. We follow Rackoff [43] and proceed by an induction on the number of relevant counters that can be pumped. Let r be the number of relevant counters. We define a function $f : [0, r-1] \to \mathbb{N}$ so that $f(l)$ is a bound on the maximal length of shortest runs r^X_{up} and r^X_{dn} that pump the relevant counters in X and ignore the remaining relevant counters by setting them to ω. The bound is taken over all initial markings, and we show in the proof that it exists. Moreover, it is taken over

all sets X with $|X| \leq l$. Note that we are only interested in sets that ignore at least one relevant counter. This is to match the definition of r^i_{up} and r^j_{dn}. Clearly, $f(0) = 1$, if there are no counters to pump, the empty run works. In the induction step, assume we want to pump a set X of $l+1$ counters. The set of markings that enable pumping runs is upward-closed. By the well-quasi-order on $\mathbb{N}^d$, it contains a finite set of minimal elements. We show how to compute the minimal elements. Once we have them, we also have the corresponding runs, and hence a bound on $f(l+1)$.

We compute the set of minimal markings that admit pumping runs for X. We do have pumping runs from the marking with value k in all counters from X, namely $r^{\mathbb{Z}}_{up}$ and $r^{\mathbb{Z}}_{dn}$. For every marking that is strictly smaller than a marking we already have, we check the existence of pumping runs using the Karp-Miller tree, and we can use the tree to construct a run if the answer is positive. We can rely on the computability of reach and breach by $\mathsf{(b)}$, there is at least one relevant counter we ignore. The challenge lies in the configurations that are incomparable to the minimal ones we already have. Consider a configuration that is strictly smaller than a minimal one in the counters Y. We set the remaining counters to ω, and check in the Karp-Miller tree the existence of up- and down-pumping runs. If the answer is positive, the induction hypothesis applies and gives us runs of length at most $f(|Y|)$. These runs may not be pumping on $X \setminus Y$, and even have a negative effect there. We construct pumping runs for the full set X with the technique from Lemma 13. The lemma gives us a bound for the values of the counters in $X \setminus Y$ that is needed to enable the constructed runs, $k \cdot (f(|Y|)+1)$. Note how the bound on the length of the runs from the induction hypothesis is crucial to obtain the bound on the counter values.

Armed with Lemma 13, we return to the original goal of computing $\mathsf{reach}(v, A)$. A clever trick leads to the following lemma.

▶ **Lemma 14.** *Let N be perfect except for (R3) and assume $\neg\mathsf{(b)} \wedge \neg\mathsf{(c)} \wedge \neg\mathsf{(d)}$ holds. We can compute a bound Bd so that $\mathsf{reach}(v, A)$ is computable for all markings v with $v[i] > Bd$ for some i.*

Proof Sketch. Let H be the set of minimal non-zero solutions to the homogeneous characteristic system, and let $M = \max_{h \in H} \|h\|$ be a bound on the norm of any such minimal solution (this is at most exponential by standard ILP arguments). Let Bd' be the bound given by Lemma 13, so if some input and output counter is larger than Bd', then N is perfect. The algorithm returns $Bd = Bd' \cdot d \cdot M$, where d is the number of counters.

The reasoning is as follows. Let v fulfill $v[i] > Bd$ for some i. Similar to the definition of $\mathsf{reach}(v, A)$, let $\mathit{reach}_{\mathbb{Z}}(v, A)$ be the set of ideals of the downward closure of the set of $\mathbb{Z}$-reachable markings. For all ω-markings $v_{\mathbb{Z}} \in \mathit{reach}_{\mathbb{Z}}(v, A)$, we show how to compute a set of ideals $I_{v_{\mathbb{Z}}} \subseteq \mathbb{N}^d_\omega$. This solves problem since $\mathsf{reach}(v, A) = \bigcup_{v_{\mathbb{Z}} \in \mathit{reach}_{\mathbb{Z}}(v,A)} I_{v_{\mathbb{Z}}}$ will hold.

To construct $I_{v_{\mathbb{Z}}}$, we use $v_{\mathbb{Z}}$ as the output marking. We have $\Omega(v) \subseteq \Omega(v_{\mathbb{Z}}) \subseteq \Omega(\mathit{out}(A))$: the first containment holds as $\mathit{reach}_{\mathbb{Z}}$ can only add ω's, and the second holds as $\mathit{reach}_{\mathbb{Z}}$ has to respect the output marking. We perform a case distinction. If $\Omega(v) \subsetneq \Omega(v_{\mathbb{Z}})$ is a strict inclusion, and so $v_{\mathbb{Z}}$ has additional ω entries compared to v, we can pick any $j \in \Omega(v_{\mathbb{Z}}) \setminus \Omega(v)$ and apply Lemma 13: the NGVAS is perfect, and hence $v_{\mathbb{Z}}$ is in fact reachable. We therefore output $I_{v_{\mathbb{Z}}} = \{v_{\mathbb{Z}}\}$.

The difficult case is that $\Omega(v) = \Omega(v_{\mathbb{Z}})$. In this case we want to use the definition of Bd to argue that some counter j fulfills $v_{\mathbb{Z}}[j] > Bd'$, in which case we can apply Lemma 13 as above. We argue as follows. Since $v_{\mathbb{Z}}$ is $\mathbb{Z}$-reachable, $(v, v_{\mathbb{Z}})$ is a (projection of a) solution to the characteristic system. Hence, we can write $(v, v_{\mathbb{Z}}) = \sum_{k=1}^{n} h_k$ as a sum of minimal homogeneous solutions. Here is a crucial trick, every h_k must contribute to $v_{\mathbb{Z}}$: when

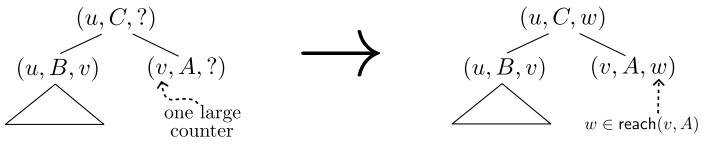
considering $h_k \in \mathbb{N}^{2d}$ as vector of length $2d$, we must have $h_k[(d+1) : 2d] \neq 0^d$. Otherwise, by removing h_k from the sum and using monotonicity, we can create a larger $\mathbb{Z}$-reachable marking $v'_{\mathbb{Z}}$, contradiction the maximality of $v_{\mathbb{Z}}$.

Since $\|h_k\| \leq M$ for all k by the definition of M, and $\|v\| \geq v[i] > Bd' \cdot d \cdot M$, it follows that the length n of the sum fulfills $n > Bd' \cdot d$. Next consider $K_j = \{k \in \{1, \ldots, n\} \mid h_k[j] \neq 0\}$ for all $j \notin \Omega(v)$. As argued above, $\bigcup_{j \notin \Omega(v)} K_j = \{1, \ldots, n\}$, every h_k contributes to a set K_j. Since we have at most d sets K_j and their union contains $n > Bd' \cdot d$ elements, by the pigeonhole principle we have $|K_j| > Bd'$ for some j. This implies $v_{\mathbb{Z}}[j] > Bd'$, counter j is large in the output. We now fulfill the assumption to apply Lemma 13 and, similar to the other case, obtain that $v_{\mathbb{Z}}$ is in fact reachable from v. Hence again we set $I_{v_{\mathbb{Z}}} = \{v_{\mathbb{Z}}\}$. ◀

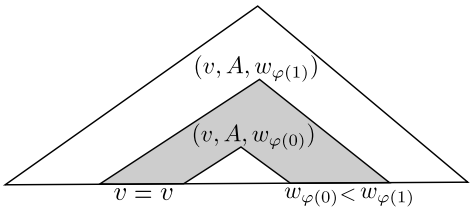
7.5 Computing (b)reach: Hard Case 2

It remains to compute $\mathsf{reach}(v, A)$ for markings v in which all relevant counters are bounded by Bd. The idea is to conduct an exhaustive search. The objects computed by this search are so-called *marked parse trees*, parse trees in the context-free grammar underlying the NGVAS whose nodes are decorated by input and output markings (v, σ, w), very similar to how the NGVAS annotates the terminals and non-terminals by input and output markings. There are two techniques that guarantee the termination of our search.

The first technique is that once we encounter a node whose input marking v exceeds the bound Bd in some counter, as illustrated below, we do not further expand the node but rely on the computability of $\mathsf{reach}(v, A)$ in Lemma 14, instead. This, however, is not enough for termination.



The second technique is an acceleration that introduces ω entries when an output marking is detected to grow unboundedly. Note that we cannot use the same acceleration as in the Karp-Miller tree. There, our goal was to find ω entries in the middle of the derivation. Here, our goal is to find the ω entries that are the result of the derivation. To motivate our notion of acceleration, we first consider a version of the search without it. We construct a search tree whose nodes are labeled by marked parse trees. A child node expands the parse tree of its parent, meaning we have a bottom-up construction and the parent-child relationship is one of being a subtree. Since we are only interested in which outputs are possible for a given input marking and non-terminal, we disallow trees where a node and a successor share the same label. Now, assume the first technique does not apply. Then the set of possible input markings is finite, and there are only finitely many marked parse trees for each height. As a result, the search tree has finite outdegree. Hence, if the search does not terminate, it contains an infinite path. We consider the labeling of the root nodes $(v_i, A_i, w_i)_{i \in \mathbb{N}}$ in the marked parse trees on this path. As there are finitely many input markings and non-terminals, the sequence contains an infinite subsequence of the form $(v, A, w_{\varphi(i)})_{i \in \mathbb{N}}$. By the well-quasi-order of $\mathbb{N}_\omega^d$ and the fact that the output markings grow unboundedly, we can even assume $w_{\varphi(0)} < w_{\varphi(1)} < \ldots$ This is precisely where we detect an acceleration. Like in the Karp-Miller tree, we introduce an ω in the output counters where the inequality is strict. The acceleration situation is illustrated below.



We can apply the acceleration at most d times along a branch. Since marked parse trees are not allowed to repeat nodes along a branch, we have termination.

To prove the soundness of the acceleration, observe that the derivation from $(v, A, w_{\varphi(1)})$ to $(v, A, w_{\varphi(0)})$ can be repeated. The fact that the initial marking coincides and the final markings satisfy $w_{\varphi(0)} < w_{\varphi(1)}$, combined with the monotonicity of firing in VAS, guarantees that the repetition is enabled.

▶ **Lemma 15.** *Let N be perfect except for (R3) and assume $\neg$(b) $\wedge$ $\neg$(c) $\wedge$ $\neg$(d) holds. Then* $\mathsf{reach}(v, A)$ *is computable for all markings $v \leq Bd^d$.*

8 Complexity

It remains to define the rank of an NGVAS and analyze the complexity of our decision procedure. For the latter, we employ a precise analysis of recursive procedures, which we believe will be of independent interest.

8.1 Bad Partially Nested Sequences

Recursions with ranks from $\mathbb{N}^\beta$ for $\beta \in \mathbb{N}$ are common in the context of infinite-state systems, and there are generic tools for analyzing their complexity: *controlled bad sequences* for tail recursions [16] and *controlled bad nested sequences* for full (non-tail) recursions [36]. Theorem VI.1 in [36] gives Hyper-Ackermann time bounds for full recursions. There is, however, a problem with these generic tools: they analyze all procedures equally but do not consider the case where some procedures are tail-recursive and others are fully recursive. To take into account this distinction, we introduce controlled bad *partially nested* sequences, a subclass of controlled bad nested sequences that admit a finer complexity analysis.

We explain the programming model that we use for our complexity analysis. The essential characteristic of our decision procedure is that (i) it is heavily recursive but (ii) the recursive functions all have the same type, they map NGVAS to sets of NGVAS. To capture this, we define a recursive program as a finite set of functions P that all have type $\mathsf{U} \to \mathbb{P}(\mathsf{U})$ for some domain U. During its computation, each function $f \in P$ may transform the input, meaning it has access to a state from U. Moreover, the function may maintain an auxiliary state from some domain A_f. An example for such auxiliary state is the Karp-Miller tree. We do not need to be precise about the syntax of programs and can also be rough about the semantics. In fact, even the domains we just introduced will be abstracted away by ranking functions. This high level of abstraction is needed to deal with objects as complicated as NGVAS and functions as complicated as reach. We only need stack frames $y = (u, f, a)$ which consist of state $u \in \mathsf{U}$, the current function $f \in P$, and auxiliary state $a \in \mathsf{A}_f$. A stack is then a finite sequence of stack frames $s = y_0 \dots y_k$ with y_k being topmost.

We are interested in proving the termination of such recursive programs. To this end, we assume to have a ranking function $r : \mathsf{U} \to \mathsf{X}$ that maps the state from U into a well-founded total order X. With U being NGVAS, the function r will define their rank. Moreover, for every $f \in P$, we need a ranking function $r_f : \mathsf{A}_f \to \mathsf{X}_f$ that maps the auxiliary state into

another well-founded total order. The auxiliary ranking function r_f may be chosen depending on $f \in P$, but we expect that all functions in the program agree on the main rank r. Finally, we assume a total order $\prec$ on the functions in P.

The goal of our complexity analysis is to separate recursive from tail-recursive behavior. We achieve this by constraining the transition relation $s_1 \to s_2$ between stacks, and analyzing computations that adhere to these constraints. An invariant of our transitions is that they do not increase the main rank. Moreover, recursive calls have to (i) reduce the main rank or (ii) call a function that is smaller in the total order $\prec$. The consequence is that transitions which only reduce the auxiliary rank cannot be recursive calls: computation on the auxiliary state is tail-recursive. We admit the following.

1. A tail-recursive step $s.(u, f, a) \to s.(v, f, b)$ and a return step from a recursive function $s.(u, f, a).(u', f', a') \to s.(v, f, b)$, provided the main rank does not go up and a rank goes down: $r(u) \geq r(v) \wedge (r(u) > r(v) \vee r_f(a) > r_f(b))$.
2. A recursive call $s.(u, f, a) \to s.(u, f, a).(v, g, b)$, provided the main rank does not go up, and the main rank goes down or the callee is smaller, $r(u) \geq r(v) \wedge (r(u) > r(v) \vee f \succ g)$.

Note that the auxiliary state can be chosen freely if we can ensure the main rank goes down.

It is readily checked that from every initial stack every computation in our model terminates. The reason is that a computation is a sequence in $\mathsf{X} \times P \times \bigcup_f \mathsf{X}_f$ that is strictly decreasing in the lexicographic ordering, and this ordering is well-founded. This is enough to show termination of our decision procedure.

To estimate the complexity, we need notions from fast-growing complexity. By further abstracting from stack frames and even stacks, we arrive at the nested sequences from [36]. Towards this abstraction, note that our termination analysis does not need the domains U and A_f, but only the well-founded total orders. We fix them to $(\mathbb{N}^\beta, \leq_{lex})$. Then (the abstract version of) a stack frame is an element of $\mathsf{Y} = \mathbb{N}^{\beta_1} \times P \times \mathbb{N}^{\beta_2}$. We also abstract away the stack content and only keep the topmost stack frame and the height of the stack. With this, (the abstract version of) a stack is an element $(y, h) \in \mathsf{Y} \times \mathbb{N}$. The abstraction of a computation is a *nested sequence*, a sequence of stacks $(y_0, h_0).(y_1, h_1)\ldots$ whose height changes as expected, $h_{k+1} \in h_k + \{-1, 0, 1\}$ for all $k \in \mathbb{N}$. To capture the requirement that every transition decreases the rank, we first define the contrary, the moment of termination. A nested sequence is *good* [36], if there are indices $k < m$ so that $y_k \leq_{lex} y_m$ and $h_{k+1}, \ldots, h_m \geq h_k$. It is *bad* if it is not good.

Our complexity analysis determines a bound on the length of bad nested sequences. This also yields a bound on the termination time for our programming model, in particular for perf, as follows. For every computation $\tau = s_0 \to s_1 \to \ldots$ in a program, the abstraction $\alpha(\tau) = (y_0, h_0).(y_1, h_1)\ldots$ is a bad nested sequence.

Clearly, without any restrictions even non-nested bad sequences like $(1, 0) \to (0, B) \to (0, B-1) \to \ldots$ in $\mathbb{N}^2$ may be arbitarily long. Hence we control the way in which transitions may modify the ranks [16]. Let $c\colon \mathbb{N} \to \mathbb{N}$ be a strictly increasing (control) function and let $m \in \mathbb{N}$ be a bound on the size of the initial stack. The nested sequence $(y_0, h_0).(y_1, h_1)\ldots$ is (c, m)-*controlled* [36] if $\|y_k\| \leq c^{(k)}(m)$ for all $k \in \mathbb{N}$. Here, we use $c^{(k)}$ for the k-fold application of c. Note that the boundedness of the stack frames is not built into our programming model. So to employ controlled sequences for the complexity analysis, we need to argue that perf respect the bounds. This, however, will be unproblematic as we will admit a general primitive-recursive control c.

Recall that $\mathsf{Y} = \mathbb{N}^{\beta_1} \times P \times \mathbb{N}^{\beta_2}$. For controlled bad *nested* sequences, the only upper bound on the length in the literature is $\mathfrak{F}_{|P|\cdot\omega^{\beta_1+\beta_2}}$, obtained from [36, Theorem VI.1], which assumes that $c(m) = m + 1$. Here, $\mathfrak{F}_\alpha$ is the hierarchy of fast-growing functions, see e.g. [48]

for details. With our analysis, transitions are allowed to take an arbitrary primitive-recursive amount of time, and we can improve the upper bound to $\mathfrak{F}_{\omega^{\beta_1}}$. Observe that this eliminates the dependence on β_2 and on $|P|$.

What allows us to improve the analysis is the fact that, in our programming model, computation on the auxiliary rank is tail-recursive. Note that this is not yet included in the above definition of nested sequences.

▶ **Definition 16.** *A nested sequence* $(y_0, h_0).(y_1, h_1)\ldots$ *is* partially nested *if* $y_k = (x, f, z_k)$ *and* $y_{k+1} = (x, f, z_{k+1})$ *implies* $h_{k+1} \leq h_k$*, for all* $k \in \mathbb{N}$*.*

We are allowed to analyze the length of controlled bad partially nested sequences, because the abstraction $\alpha(\tau)$ of a computation τ in our model is not only a bad nested sequence but actually a bad partially nested sequence. The bound we obtain with this analysis will also be precise, as we have the following. For every bad partially nested sequence $\sigma = (y_0, h_0).(y_1, h_1)\ldots$ there is a computation $\tau = s_0 \to s_1 \to \ldots$ so that $\sigma = \alpha(\tau)$.

We explain how partially nested sequences eliminate the dependence on β_2 and $|P|$. Consider the case where $\beta_1 = 1$ and so $\mathsf{Y} = \mathbb{N} \times P \times \mathbb{N}^{\beta_2}$. Assume we start with some value $k \in \mathbb{N}$ for the main rank. Then a partially nested sequence can be modelled as a tail recursion with a rank in $\mathbb{N}^{(k+1)|P|\cdot\beta_2}$, with the following argument. At every moment in the sequence, the call stack has height at most $(k+1)|P|$. We can store the $(k+1)|P|$ auxiliary ranks that lie on the stack in a long tuple. Using the bad sequence analysis of [16, Prop. 5.2], we arrive at a complexity of $\mathfrak{F}_{(k+1)|P|\cdot\beta_2} \subseteq \mathfrak{F}_\omega$. In fact, already $\beta_1 = 1 = \beta_2 = |P|$ would lead to $\mathfrak{F}_\omega$, as one can directly implement the Ackermann function. The consequence is that the details β_2 and $|P|$ vanish once we round up to the next power of ω, which will always be ω^{β_1}.

▶ **Theorem 17.** *Let* $\beta_1, m, \beta_2 \in \mathbb{N}$ *and let* $|P|$ *be finite. Let* α *be an ordinal and let* $c \in \mathfrak{F}_\alpha$ *be strictly increasing. Then the length of* (c, m)*-controlled bad partially nested sequences over* $\mathbb{N}^{\beta_1} \times P \times \mathbb{N}^{\beta_2}$ *is in* $\mathfrak{F}_{\alpha+\omega^{\beta_1}}$*.*

8.2 Rank of NGVAS

To analyze the complexity of our decision procedure using Theorem 17, we have to define a main rank (from $\mathbb{N}^{\beta_1}$) and an auxiliary rank (from $\mathbb{N}^{\beta_2}$). Recall that the main rank is the one in which we admit recursion, and the auxiliary rank is for non-recursive computations. Our decision procedure is recursive in NGVAS objects: the functions expect NGVAS as input and return sets of NGVAS as output. This means the main rank from $\mathbb{N}^{\beta_1}$ will measure the depth of the recursion that is needed to analyze an NGVAS. In fact, also the first γ-components of the auxiliary rank $\beta_2 = \gamma + \delta$ contribute to the complexity of an NGVAS, but are not needed for the recursion. We now define the complexity measure from $\mathbb{N}^{\beta_1} \times \mathbb{N}^{\gamma}$, and call it the *rank of an NGVAS*.

We begin by explaining why the complexity of an NGVAS is split between the main and the auxiliary rank. Recall that an NGVAS is a nesting of (extended) context-free grammars, and every such grammar can either be non-linear or linear. The main rank $\mathbb{N}^{\beta_1}$ will (almost) only consider non-linear grammars. The auxiliary rank $\mathbb{N}^{\gamma}$ will be for linear grammars. We use the notation $\mathsf{rk}(N) = (\mathsf{rk}_{\mathsf{nl}}(N), \mathsf{rk}_{\mathsf{lin}}(N)) \in \mathbb{N}^{\beta_1} \times \mathbb{N}^{\gamma}$. Let d be the number of counters.

Main Rank $\mathsf{rk}_{\mathsf{nl}}(N)$

The main rank is

$$\mathsf{rk}_{\mathsf{nl}}(N) \;=\; (|\overline{Un}|, \mathsf{srk}(N), \mathsf{ix}(N)) \in \mathbb{N} \times \mathbb{N}^{d+1} \times \mathbb{N}\,.$$

There are three components that together yield $\beta_1 = d + 3$. We explain them starting from the most significant one.

Component $|\overline{Un}| \in \mathbb{N}$

This is the number of counters that are subject to reachability constraints. Like KLMST [38], our decomposition never removes a reachability constraint from an NGVAS. However, in Section 7 we often considered variants of the given NGVAS with smaller sets $\overline{Un}$. The component $|\overline{Un}|$ is the most important in order to ensure Lemma 10.

Component $\mathsf{srk}(N) \in \mathbb{N}^{d+1}$

This so-called system rank is inspired by the rank for KLMST sequences [38]. The main difference is that NGVAS form a branching structure. To bridge the gap, we identify the branch in the NGVAS that can lead to the deepest recursion. A *branch* of N_0 is a sequence of NGVAS $\mathsf{b} = N_0 \dots N_k$ where N_{i+1} is the child of N_i for all $i < k$. We thus have

$$\mathsf{srk}(N) \;=\; \max_{\mathsf{b} \text{ branch in } N} \mathsf{brk}(\mathsf{b}) \;.$$

The depth of the recursion that is needed to analyze a branch is the sum of the depths that are needed to analyze the NGVAS that lie on this branch:

$$\mathsf{brk}(\mathsf{b}) \;=\; \sum_{N \in \mathsf{b}} \mathsf{lrk}(N) \;.$$

The idea behind the local rank of an NGVAS is to replace the syntactic notion of counters by a semantic one, and determine the number of semantic counters that the NGVAS uses. This means the recursion that depends on the local rank removes the semantic counters. To understand the idea of semantic counters, assume the NGVAS has two (syntactic) counters and every terminal with effect $k \in \mathbb{Z}$ on the first counter has effect $2k$ on the second. Then the counter values move along $(1, 2) \cdot \mathbb{N}$, so only one semantic counter is needed. The idea of semantic counters, or the dependence among syntacic counters, is naturally captured as the dimension of a suitable vector space [38]. We adapt it to our needs.

A *cycle* in an NGVAS is a derivation $A \to^* \alpha.A.\beta$ that reproduces the non-terminal A, along with $\alpha, \beta \in \Sigma^*$. The terminals $M \in \Sigma$ are childNGVAS, and the effect $\mathsf{eff}(M) \subseteq \mathbb{Z}^d$ of such a child is the set of effects $v_{out} - v_{in}$ that stem from its linear restriction $(v_{in}, v_{out}) \in M.Rs$. This overapproximates the effects of runs in $R_{\mathbb{N}}(M)$. For a terminal sequence $\alpha = M_0 \dots M_l$, the effect is the sum $\mathsf{eff}(\alpha) = \mathsf{eff}(M_0) + \dots + \mathsf{eff}(M_l)$. We define the *left cycle space* as the vector space

$$\mathbf{V}_{\mathsf{lft}}(N) \;=\; \mathsf{span}\{v \in \mathsf{eff}(\alpha) \;\mid\; \exists A.\; A \to^* \alpha.A.\beta\} \subseteq \mathbb{R}^d.$$

Assume N is non-linear and recall that the local rank is a vector of dimension $d+1$. This vector only has one non-zero entry, namely (d minus) the dimension of the cycle space. This entry is the number of non-terminals:

$$\mathsf{lrk}(N) \;=\; 1_{d - \mathsf{dim}(\mathbf{V}_{\mathsf{lft}}(N))} \cdot |\Gamma|.$$

The subtraction moves higher dimensional spaces to the front, and in the lexicographic order $1_i > 1_j$ if $i < j$. If N is linear, $\mathsf{lrk}(N) = 0$.

The definition calls for comments. We only use the left cycle space as, for non-linear NGVAS, the left and the right cycle space coincide. Moreover, we do not need to define the dimension for each non-terminal: it will be the same due to strong connectedness. Finally, we add up the local ranks when determining the branch rank, and this will fill further dimensions with values.

Component $\mathsf{ix}(N) \in \mathbb{N}$

As we have defined it so far, the main rank is for the analysis of non-linear NGVAS. To also support linear NGVAS, we add the *local index* as the last component of the main rank. We explain the recursion on linear NGVAS that needs this index.

To decompose linear NGVAS, we fold them into VASS with nested zero tests (VASSnz) [44], and then apply the recent decomposition for VASSnz [22]. We explain both steps. Consider a linear and non-nested NGVAS. Using convolution, we fold it into an ordinary VAS with twice the number of counters, for the run on the left and the run on the right of the single non-terminal. Adding the nesting that an NGVAS has, we obtain a VASSnz [2]. The decomposition for VASSnz [22] proceeds recursively on the nesting depth. The benefit of the recursion is that it can treat each nesting level as an ordinary VASS.

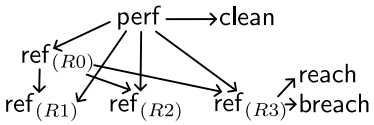
The *local index* measures the nesting depth of the VASSnz that results from an NGVAS by folding the linear components. Since linear and non-linear components may alternate in an NGVAS, the formal definition is *modulo the system rank*. This means we treat subNGVAS thar are non-linear or have lower system rank as terminals. The local index $\mathsf{ix}(N) \in \mathbb{N}$ is then the index of the resulting grammar. If N is non-linear, then $\mathsf{ix}(N) = 0$.

Auxiliary Rank $\mathsf{rk}_{\mathsf{lin}}(N)$

The definition of the local index leads to the notion of a *main branch*, which behaves like the top level VASSnz in [22]. We define the main branch mb_N to consist of all subNGVAS of N with the same system rank and the same local index as N. This is unique. The auxiliary rank $\mathsf{rk}_{\mathsf{lin}}(N) \in \mathbb{N}^\gamma$ is then the rank of the system represented by mb_N, as it would be ranked in [22]. The details can be found in the appendix.

8.3 Analyzing Our Algorithm using Theorem 17

We now apply Theorem 17 to determine a bound on the complexity of our decision procedure. The theorem expects stack frames from $\mathsf{Y} = \mathbb{N}^{\beta_1} \times P \times \mathbb{N}^{\beta_2}$. We already discussed the main rank $\mathsf{rk}_{\mathsf{nl}}(N) \in \mathbb{N}^{\beta_1}$ that controls the recursion. The recursive functions P in our decision procedure are depicted in the following graph:



An edge $f \to g$ indicates that function $f(N)$ will invoke function $g(M)$, and the NGVAS argument M it passes may still have the same main rank, $\mathsf{rk}_{\mathsf{nl}}(M) = \mathsf{rk}_{\mathsf{nl}}(N)$. This means the functions have to be ordered $f \succ g$, and so the total order on the functions that Theorem 17 expects can be an arbitrary linearization of this graph. As for the auxiliary rank $\mathbb{N}^{\beta_2}$ with $\beta_2 = \gamma + \delta$, we already defined the linear rank $\mathsf{rk}_{\mathsf{lin}}(N) \in \mathbb{N}^\gamma$. We now focus on the implementation of the functions, and the auxiliary rank from $\mathbb{N}^\delta$ they need.

Function $\mathsf{ref}_{(\mathsf{R0})}$ establishes perfectness condition $(R0)$ stating that all children are perfect. Thanks to omitted cleanness conditions, we only need to invoke this for linear NGVAS. For linear NGVAS, we do not have the guarantee that children have a smaller rank. Here, we rely on the recent VASSnz decomposition to avoid calls to perf and remain within our programming model.

Function $\mathsf{ref}_{(\mathsf{R3})}$ computes a Karp-Miller tree to check whether a pumping situation exists and, if the check fails, a coverability grammar for the decomposition, see Section 7. Both computations are non-recursive, which means the auxiliary rank $\mathbb{N}^\delta$ has to be large enough

to accomodate them. Before we get into the details, we note that both computations invoke the functions reach and breach. These functions are smaller in the total order of functions, and so we can invoke them without having to reduce the main rank.

We explain how we compute the Karp-Miller tree, for the coverability grammar we proceed similarly. We store configurations c in the tree as elements from $\mathbb{N}^{2d+2}$, with d counters for the input marking, d counters for the output marking, and 2 counters for the state as $(i, |Q| - i)$. For every branch $\mathsf{b} = c_0, \ldots, c_r$ in the tree, we define a rank, namely $|\mathbb{N}^{2d+2} \setminus \uparrow\{c_0, \ldots, c_r\}|$. This is the size of the downward-closed set of configurations which may still be added to the branch. Using the notion of *types* [16], we can represent the size of a downward-closed set $D \subseteq \mathbb{N}^{2d+2}$ as an element of $(\mathbb{N}^{2d+2}, \leq_{lex})$. For this, we write the downward-closed set D as a union of ω-markings/copies of $\mathbb{N}^{n_i}$, i.e. $D = \bigcup_{i=1}^{r} \mathbb{N}^{n_i}$. Now $\mathsf{type}(D) \in \mathbb{N}^{2d+2}$ counts the number of ω-markings with a given number k of ω-entries: $\mathsf{type}(D)[k] = |\{i \mid 2d+2-n_i = k\}|$. For example, $\downarrow(1, \omega) = (0, \omega) \cup (1, \omega)$ has $\mathsf{type}(\downarrow(1, \omega))[0] = 0$ and $\mathsf{type}(\downarrow(1, \omega))[1] = 2$. To simplify the notation, we just write $\mathsf{type}(\mathsf{b})$ for $\mathsf{type}(\mathbb{N}^{2d+2} \setminus \uparrow\mathsf{b})$.

The auxiliary rank in $\mathbb{N}^{\delta}$ that we need for the tree is $(\max_{\mathsf{b}} \mathsf{type}(\mathsf{b}),$ #branches in current round$) \in \mathbb{N}^{2d+3}$. We build the tree in breadth-first fashion, and the last component is a loop counter, like above.

It remains to compute reach and breach. In the simple cases, and similar to $\mathsf{ref}_{(\mathsf{R0})}$, we can use a recursion that calls perf. This is justified because $|\overline{Un}|$ decreased. For Hard Case 1, we have to compute $f(r-1)$ with r being the number of relevant counters. We have an outer loop for r, and an inner loop that maintains the currently known minimal markings fulfilling the pumping property. Moreover, we construct several Karp-Miller trees, but reuse the space once we have the result. Finally, we obtain Bd using integer linear programming. For Hard Case 2, we compute marked parse trees similar to Karp-Miller trees.

9 Conclusion

We presented a decision procedure for the reachability problem in pushdown vector addition systems. It relies on several insights that may be of independent interest. The first insight is a wide tree theorem for context-free grammars that guarantees the existence of derivation trees of height logarithmic in the length of the word, at the cost of only preserving Parikh equivalence. The second insight is the notion of vertical pumping which, intuitively, harmonizes the manipulation of the stack and the counters. The third insight is that Rackoff's induction generalizes beyond coverability. The fourth insight is that the problem is best reduced to itself, with an input that is smaller in a well-founded order. The last insight is that a combination of recursive and tail-recursive functions over ordinals admits a better complexity analysis than what was known in the literature on fast-growing complexity.

References

1 Ashwani Anand, Sylvain Schmitz, Lia Schütze, and Georg Zetzsche. Verifying unboundedness via amalgamation. In *LICS*, pages 4:1–4:15. ACM, 2024. doi:10.1145/3661814.3662133.

2 Mohamed Faouzi Atig and Pierre Ganty. Approximating petri net reachability along context-free traces. In *FSTTCS*, volume 13 of *LIPIcs*, pages 152–163. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.152.

3 Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. Checking refinement of asynchronous programs against context-free specifications. In *ICALP*, volume 261 of *LIPIcs*, pages 110:1–110:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.110.

4 Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. Context-bounded verification of context-free specifications. *Proc. ACM Program. Lang.*, 7(POPL):2141–2170, 2023. doi:10.1145/3571266.

5 Clotilde Bizière and Wojciech Czerwinski. Reachability in one-dimensional pushdown vector addition systems is decidable. In *STOC*, pages 1851–1862. ACM, 2025. doi:10.1145/3717823.3718149.

6 Michael Blondin and François Ladouceur. Population protocols with unordered data. In *ICALP*, volume 261 of *LIPIcs*, pages 115:1–115:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.115.

7 Rémi Bonnet. The reachability problem for vector addition system with one zero-test. In *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 145–157. Springer, 2011. doi:10.1007/978-3-642-22993-0_16.

8 Rémi Bonnet. *Theory of Well-Structured Transition Systems and Extended Vector-Addition Systems*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, 2013. URL: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/bonnet-phd13.pdf.

9 Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Separability of reachability sets of vector addition systems. In *STACS*, volume 66 of *LIPIcs*, pages 24:1–24:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.24.

10 Lorenzo Clemente, Slawomir Lasota, Ranko Lazic, and Filip Mazowiecki. Timed pushdown automata and branching vector addition systems. In *LICS*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005083.

11 Wojciech Czerwiński, Ismaël Jecker, Sławomir Lasota, Jérôme Leroux, and Łukasz Orlikowski. New lower bounds for reachability in vector addition systems. In *FSTTCS*, volume 284 of *LIPIcs*, pages 35:1–35:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.FSTTCS.2023.35.

12 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. In *STOC*, pages 24–33. ACM, 2019. doi:10.1145/3313276.3316369.

13 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is ackermann-complete. In *FOCS*, pages 1229–1240. IEEE, 2021. doi:10.1109/FOCS52979.2021.00120.

14 Stéphane Demri, Marcin Jurdzinski, Oded Lachish, and Ranko Lazic. The covering and boundedness problems for branching vector addition systems. *J. Comput. Syst. Sci.*, 79(1):23–38, 2013. doi:10.1016/J.JCSS.2012.04.002.

15 J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundam. Informaticae*, 31(1):13–25, 1997. doi:10.3233/FI-1997-3112.

16 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with dickson's lemma. In *LICS*, pages 269–278. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.39.

17 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for wsts, part I: completions. In *STACS*, volume 3 of *LIPIcs*, pages 433–444. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2009. doi:10.4230/LIPIcs.STACS.2009.1844.

18 Alain Finkel, Jérôme Leroux, and Grégoire Sutre. Reachability for two-counter machines with one test and one reset. In *FSTTCS*, volume 122 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.31.

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

20 R. Guttenberg, E. Keskin, and R. Meyer. Pvass reachability is decidable, 2026. arXiv:2504.05015.

21 Roland Guttenberg. Flattability of priority vector addition systems. In *ICALP*, volume 297 of *LIPIcs*, pages 141:1–141:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ICALP.2024.141.

22 Roland Guttenberg, Wojciech Czerwinski, and Slawomir Lasota. Reachability and related problems in vector addition systems with nested zero tests. In *LICS*, pages 581–593. IEEE, 2025. doi:10.1109/LICS65433.2025.00050.

23 Peter Habermehl, Roland Meyer, and Harro Wimmel. The downward-closure of petri net languages. In *ICALP (2)*, volume 6199 of *Lecture Notes in Computer Science*, pages 466–477. Springer, 2010. doi:10.1007/978-3-642-14162-1_39.

24 Piotr Hofman, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, Sylvain Schmitz, and Patrick Totzke. Coverability trees for petri nets with unordered data. In *FoSSaCS*, volume 9634 of *Lecture Notes in Computer Science*, pages 445–461. Springer, 2016. doi:10.1007/978-3-662-49630-5_26.

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

26 Eren Keskin and Roland Meyer. On the separability problem of VASS reachability languages. In *LICS*, pages 49:1–49:14. ACM, 2024. doi:10.1145/3661814.3662116.

27 S. Rao Kosaraju. Decidability of reachability in vector addition systems. In *STOC*, pages 267–281. ACM, 1982.

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

29 R. Lazic. The reachability problem for vector addition systems with a stack is not elementary. *CoRR*, abs/1310.1767, 2013. arXiv:1310.1767.

30 Ranko Lazic. The reachability problem for branching vector addition systems requires doubly-exponential space. *Inf. Process. Lett.*, 110(17):740–745, 2010. doi:10.1016/J.IPL.2010.06.008.

31 Ranko Lazic and Sylvain Schmitz. The complexity of coverability in ν-petri nets. In *LICS*, pages 467–476. ACM, 2016. doi:10.1145/2933575.2933593.

32 Jérôme Leroux. The general vector addition system reachability problem by presburger inductive invariants. In *LICS*, pages 4–13. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.10.

33 Jérôme Leroux. Vector addition system reachability problem: A short self-contained proof. In *LATA*, volume 6638 of *Lecture Notes in Computer Science*, pages 41–64. Springer, 2011. doi:10.1007/978-3-642-21254-3_3.

34 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *FOCS*, pages 1241–1252. IEEE, 2021. doi:10.1109/FOCS52979.2021.00121.

35 Jérôme Leroux, M. Praveen, Philippe Schnoebelen, and Grégoire Sutre. On functions weakly computable by pushdown petri nets and related systems. *Log. Methods Comput. Sci.*, 15(4), 2019. doi:10.23638/LMCS-15(4:15)2019.

36 Jérôme Leroux, M. Praveen, and Grégoire Sutre. Hyper-ackermannian bounds for pushdown vector addition systems. In *CSL-LICS*, pages 63:1–63:10. ACM, 2014. doi:10.1145/2603088.2603146.

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

38 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *LICS*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785796.

39 Jérôme Leroux and Grégoire Sutre. Reachability in two-dimensional vector addition systems with states: One test is for free. In *CONCUR*, volume 171 of *LIPIcs*, pages 37:1–37:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.37.

40 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *ICALP (2)*, volume 9135 of *Lecture Notes in Computer Science*, pages 324–336. Springer, 2015. doi:10.1007/978-3-662-47666-6_26.

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

42 T. Place and M. Zeitoun. Separating regular languages with first-order logic. In *CSL-LICS*, pages 75:1–75:10. ACM, 2014. doi:10.1145/2603088.2603098.

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

44 Klaus Reinhardt. Reachability in petri nets with inhibitor arcs. In *RP*, volume 223 of *Electronic Notes in Theoretical Computer Science*, pages 239–264. Elsevier, 2008. doi:10.1016/J.ENTCS.2008.12.042.

45 Thomas W. Reps, Susan Horwitz, and Shmuel Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL*, pages 49–61. ACM Press, 1995. doi:10.1145/199448.199462.

46 Fernando Rosa-Velardo and David de Frutos-Escrig. Decidability and complexity of petri nets with unordered data. *Theor. Comput. Sci.*, 412(34):4439–4451, 2011. doi:10.1016/J.TCS.2011.05.007.

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

48 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.

49 Sergey Sevast'janov and Wojciech Banaszczyk. To the Steinitz Lemma in Coordinate Form. *Discrete Mathematics*, 169(1-3):145–152, 1997. doi:10.1016/0012-365X(94)00240-J.

50 M Sharir and A Pnueli. *Two approaches to interprocedural data flow analysis*. New York Univ. Comput. Sci. Dept., 1978.

51 Ernst Steinitz. Bedingt Konvergente Reihen und Konvexe Systeme, 1913.

52 Kumar Neeraj Verma and Jean Goubault-Larrecq. Karp-miller trees for a branching extension of VASS. *Discret. Math. Theor. Comput. Sci.*, 7(1):217–230, 2005. doi:10.46298/DMTCS.350.

53 Georg Zetzsche. *Monoids as Storage Mechanisms*. PhD thesis, Kaiserslautern University of Technology, Germany, 2016. URL: https://kluedo.ub.rptu.de/frontdoor/index/index/docId/4400.