Reachability in Bidirected Pushdown VASS

Moses Ganardi
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Rupak Majumdar
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Andreas Pavlogiannis
Aarhus University, Denmark

Lia Schütze
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Georg Zetzsche
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

A pushdown vector addition system with states (PVASS) extends the model of vector addition systems with a pushdown store. A PVASS is said to be *bidirected* if every transition (pushing/popping a symbol or modifying a counter) has an accompanying opposite transition that reverses the effect. Bidirectedness arises naturally in many models; it can also be seen as a overapproximation of reachability. We show that the reachability problem for *bidirected* PVASS is decidable in Ackermann time and primitive recursive for any fixed dimension. For the special case of one-dimensional bidirected PVASS, we show reachability is in PSPACE, and in fact in polynomial time if the stack is polynomially bounded. Our results are in contrast to the *directed* setting, where decidability of reachability is a long-standing open problem already for one dimensional PVASS, and there is a PSPACE-lower bound already for one-dimensional PVASS with bounded stack.

The reachability relation in the bidirected (stateless) case is a congruence over $\mathbb{N}^d$. Our upper bounds exploit saturation techniques over congruences. In particular, we show novel elementary-time constructions of semilinear representations of congruences generated by finitely many vector pairs. In the case of one-dimensional PVASS, we employ a saturation procedure over bounded-size counters.

We complement our upper bound with a TOWER-hardness result for arbitrary dimension and k-EXPSPACE hardness in dimension $2k + 6$ using a technique by Lazić and Totzke to implement iterative exponentiations.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Vector addition systems, Pushdown, Reachability, Decidability, Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.124

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

Related Version *Full Version*: `https://arxiv.org/abs/2204.11799` [20]

1 Introduction

The reachability problem for infinite-state systems is one of the most basic and well-studied tasks in verification. Given an infinite-state system and two configurations c_1 and c_2 in the system, it asks: Is there a run from c_1 to c_2? *Pushdown systems* (PDS) and *vector addition systems with states* (VASS) are prominent models for which the reachability problem has been studied extensively. Each of them features a finite set of control states and a storage mechanism that holds an unbounded amount of information. In a PDS, there is a stack where we can push and pop letters. In a VASS, there is a set of counters which can be *incremented* and *decremented*, but not tested for zero. Reachability in both models is understood in isolation [5, 11, 37, 12, 38], but the reachability problem for their *combination* is a long-standing open problem.

Pushdown VASS. A *pushdown VASS* (PVASS) combines PDS and VASS. A PVASS consists of finitely many control states and has access to both a pushdown stack (as in PDS) and counters (as in VASS). A PVASS is *d-dimensional* if it has d counters. A PVASS is a natural combination of the simple building blocks of PDS and VASS. The reachability problem for PVASS has remained a long-standing open problem [39, 50, 15], even if we combine a pushdown with a single counter.

Bidirectedness. A step toward deciding reachability is to first study natural relaxations of the reachability relation. A relaxation that has recently attracted attention is *bidirectedness*. Bidirectedness assumes that for each transition from state p to q in our infinite-state system, there exists a transition from q to p with opposite effect. For example, in bidirected pushdown systems, for each transition from p to q pushing γ on the stack, there is a transition from q to p that pops γ. Likewise, in bidirected VASS, if there is a transition from p to q that adds some vector $\boldsymbol{v} \in \mathbb{Z}^d$ to counters, then there is a transition from q to p adding $-\boldsymbol{v}$. It turns out that several tasks in program analysis can be formulated or practically approximated as reachability in bidirected pushdown systems [8, 54] or bidirected multi-pushdown systems [51, 52, 55, 40, 41, 31]. Bidirected systems have also been considered in algorithmic group theory as an algorithmic framework to provide simple algorithms for the membership problem in subgroups [42].

Reachability in bidirected systems is usually considerably more efficient than in the general case. In bidirected pushdown systems, reachability can be solved in almost linear time [8] whereas a truly subcubic algorithm for the general case is a long-standing open problem [26, 9]. Reachability in bidirected VASS is equivalent to the uniform word problem in finitely presented commutative semigroups, which is $\mathsf{EXPSPACE}$-complete [43]. A separate polynomial time algorithm for bidirected two-dimensional VASS was given in [41]. Moreover, recent results on reachability in bidirected valence systems shows complexity drops across a large variety of infinite-state systems [21]: For almost every class of systems studied in [21], the complexity of bidirected reachability is lower than in the general case (the only exception being pushdown systems, where the complexity is P-complete in both settings). For example, reachability in bidirected $\mathbb{Z}$-VASS, and even in bidirected $\mathbb{Z}$-PVASS, is in P [21].

However, little is known about bidirected PVASS. They have recently been studied in [31], where decidability of reachability in *dimension one is shown.* However, as in the non-bidirected case, decidability of reachability in bidirected PVASS is hitherto not known.

Contributions. We show that in bidirected PVASS (of arbitrary dimension), reachability is decidable. Moreover, we provide an Ackermann complexity upper bound, and show that in any fixed dimension, reachability is primitive recursive.

▶ **Theorem 1.1.** *Reachability in bidirected pushdown VASS is in* $\mathsf{ACKERMANN}$*, and primitive recursive (in* F_{4d+11}*) if the dimension* d *is fixed.*

Here, $(\mathsf{F}_\alpha)_\alpha$ is an ordinal-indexed hierarchy of fast-growing complexity classes [48], including $\mathsf{F}_3 = \mathsf{TOWER}$ and $\mathsf{F}_\omega = \mathsf{ACKERMANN}$. The formal definition of the hierarchy can be found in Section 4.3. A recurring theme in our upper bounds is that saturation techniques, the standard method to analyze pushdown systems, combine surprisingly well with counters in the bidirected setting. Saturation is used in each of our upper bounds. In Section 3, we begin the exposition with a short, self-contained proof that reachability is decidable in bidirected PVASS. It shows that non-reachability is always certified by an inductive invariant of a particular saturation procedure. In Section 4, we show the Ackermann upper bound. Here, we saturate a congruence relation that encodes the reachability relation. The upper bound relies on two key ingredients. First, we use results about Gröbner bases of polynomial ideals to show that in elementary time, one can construct a Presburger formula for the congruence generated by finitely many vector pairs. This construction serves as one step in the saturation. To show termination in Ackermannian time, we rely on a technique from [16] to bound the length of strictly ascending chains of upward closed sets of vectors. Here, the difficulty is to transfer this bound from chains of upward closed sets to chains of congruences.

In Section 5 we present a PSPACE algorithm for bidirected PVASS in dimension one.

▶ **Theorem 1.2.** *Reachability in 1-dimensional bidirected pushdown VASS is in* PSPACE*.*

Here, we rely on an observation from [31] that reachability in bidirected one-dimensional PVASS reduces to (i) coverability in bidirected one-dimensional PVASS and (ii) reachability in one-dimensional bidirected $\mathbb{Z}$-PVASS. Since (ii) is known to be in P [21], we show that (i) can be done in PSPACE. For this, we use saturation to compute, for each state pair (p, q), three bounds on counter values that determine whether coverability holds. We show that these bounds have at most exponential absolute value, which yields a PSPACE procedure.

Finally in Section 6, we show that reachability in bidirected PVASS is TOWER-hard. For this, we adapt a technique from [33] that shows a TOWER lower bound for general PVASS.

▶ **Theorem 1.3.** *Reachability in bidirected PVASS is* TOWER*-hard, and* k-EXPSPACE*-hard in dimension* $2k + 6$*.*

Related work. The model of pushdown VASS is surrounded by extensions and restrictions of the storage mechanism for which decidability is understood, the most prominent being the recent Ackermann-completeness for reachability in VASS [11, 37, 12, 38]. If instead of the stack, we have a counter with zero tests, then reachability is still decidable [47, 4]. Here, decidability even holds if we have a zero-testable counter and one additional counter that can be reset [18, 17]. Furthermore, the extension of VASS by *nested zero tests*, where for each $i \in \{1, \ldots, d\}$, we have an instruction that tests all counters $1, \ldots, i$ for zero simultaneously, also allows deciding reachability [47, 3] and can be seen as a special case of pushdown VASS [1]. Another decidability result concerns the *coverability problem*: Here, we are given a configuration c_1 and a control state q and want to know whether from c_1, one can reach some configuration in control state q. It is known that the reachability problem for d-dimensional PVASS reduces to coverability in $(d + 1)$-dimensional PVASS, and that coverability in 1-dimensional PVASS is decidable [39]. According to [15], the latter problem is PSPACE-hard and in EXPSPACE. Furthermore, if the counters in a PVASS are allowed to go negative during a run, then we speak of an *integer PVASS* ($\mathbb{Z}$-PVASS). For these, reachability is known to be decidable [25] and NP-complete [24]. However, if we extend the model of PVASS by allowing resets on the counters, then even coverability is undecidable in dimension one [50].

For VASS, several generalizations of bidirectedness have been studied. It is EXPSPACE-complete whether given two configurations are mutually reachable [35]. Moreover, if two configurations are mutually reachable, then their distance is at most doubly exponential (linear for fixed dimension) in their size [36]. Furthermore, for cyclic VASS (where each transition can be reversed by some execution), it is known that the reachability set has a semilinear representation of at most exponential size [6]. Let us note that in the VASS/Petri net literature, sometimes [6] (but not entirely consistently [35]) the term *reversible* is used to mean bidirected. However, this clashes with the reversibility notion in dynamical systems [30].

2 Preliminaries

Vectors and semilinear sets. We denote integer vectors by bold letters $\boldsymbol{x}$. The maximum norm of $\boldsymbol{x}$ is denoted by $\|\boldsymbol{x}\|$. The i-th unit vector is denoted by $\boldsymbol{e}_i$. The componentwise order $\leq$ on $\mathbb{N}^d$ is a *well-quasi order (wqo)*, i.e. for any infinite sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots$ over $\mathbb{N}^d$ there exist $i < j$ with $\boldsymbol{x}_i \leq \boldsymbol{x}_j$. We write $\boldsymbol{x} < \boldsymbol{y}$ if $\boldsymbol{x} \leq \boldsymbol{y}$ and $\boldsymbol{x} \neq \boldsymbol{y}$. This implies that the set $\min(X)$ of minimal elements in any $X \subseteq \mathbb{N}^d$ is finite. We denote by $X\!\uparrow = \{\boldsymbol{y} \in \mathbb{N}^k \mid \exists \boldsymbol{x} \in X : \boldsymbol{x} \leq \boldsymbol{y}\}$ the *upwards closure* of X. We also write $\boldsymbol{x}\!\uparrow$ for $\{\boldsymbol{x}\}\!\uparrow$. A *congruence* on a commutative monoid $(M, +)$, for example $M = \mathbb{N}^d$, is an equivalence relation $\mathcal{Q} \subseteq M \times M$ where $(a, b) \in \mathcal{Q}$ implies $(a + c, b + c) \in \mathcal{Q}$ for all $a, b, c \in M$. We also write $a \sim_{\mathcal{Q}} b$ instead of $(a, b) \in \mathcal{Q}$.

For $X \subseteq \mathbb{N}^k$ we denote by X^* the *submonoid* generated by X. A set $L \subseteq \mathbb{N}^k$ is *linear* if it is of the form $L = \boldsymbol{b} + P^*$ for some base vector $\boldsymbol{b} \in \mathbb{N}^k$ and some finite set $P \subseteq \mathbb{N}^k$ of period vectors. Finite unions of linear sets are called *semilinear*. It is well-known that a set is semilinear if and only if it is definable in *Presburger arithmetic*, i.e. first-order logic over $(\mathbb{N}, +, \leq, 0, 1)$. Furthermore, one can effectively convert between these formats in elementary time: While defining semilinear sets in Presburger arithmetic is straightforward, for the converse we can use Cooper's quantifier elimination [10] running in triply exponential time [44], see also [23] for an excellent overview. We will confuse a semilinear S with its representation, which is either a list of base and period vectors for each linear set or a defining Presburger formula, and denote by $\|S\|$ the size of its representation.

Pushdown VASS. A *d-dimensional pushdown VASS (PVASS)* is a tuple $\mathcal{P} = (Q, \Gamma, T)$ where Q is a finite set of states, Γ is a finite stack alphabet, and $T \subseteq Q \times \mathbb{Z}^d \times \mathsf{Op}(\Gamma) \times Q$ is a finite set of transitions. Here $\mathsf{Op}(\Gamma) = \{a, \bar{a} \mid a \in \Gamma\} \cup \{\varepsilon\}$ is the set of operations on the stack. A configuration over $\mathcal{P}$ is a tuple $(q, \boldsymbol{x}, s) \in Q \times \mathbb{N}^d \times \Gamma^*$. The *one-step relation* $\to$ is the smallest binary relation on configurations such that for all $(p, \boldsymbol{v}, \alpha, q) \in T$ and $\boldsymbol{x} \in \mathbb{N}^d$ with $\boldsymbol{x} + \boldsymbol{v} \geq \boldsymbol{0}$ we have: (i) If $\alpha \in \Gamma \cup \{\varepsilon\}$ then $(p, \boldsymbol{x}, s) \to (q, \boldsymbol{x} + \boldsymbol{v}, s\alpha)$ (ii) if $\alpha = \bar{a}$ then $(p, \boldsymbol{x}, sa) \to (q, \boldsymbol{x} + \boldsymbol{v}, s)$. Its transitive-reflexive closure is denoted by $\xrightarrow{*}$. We say that $\mathcal{P}$ is *bidirected* if $(p, \boldsymbol{v}, \alpha, q) \in T$ implies $(q, -\boldsymbol{v}, \bar{\alpha}, p) \in T$ where we set $\bar{\bar{a}} = a$ for $a \in \Gamma$ and $\bar{\varepsilon} = \varepsilon$. The *reachability problem* for bidirected PVASS asks: Given a bidirected PVASS $\mathcal{P}$ and two states s, t, does $(s, \boldsymbol{0}, \varepsilon) \xrightarrow{*} (t, \boldsymbol{0}, \varepsilon)$ hold?

The counter updates $\boldsymbol{u}$ in a PVASS transition $(p, \boldsymbol{u}, q)$ can be given in either unary or binary encoding since there are logspace translations in both directions: To add a binary encoded number u to a counter we push the binary notation of u to the stack, and repeatedly decrement the stack counter while incrementing u. Since this computation is deterministic, the simulation also works for bidirected PVASS.

For the Ackermann upper bound it is convenient to use pushdown VASS with a single state. A *pushdown VAS (PVAS)* $\mathcal{P} = (\Gamma, T)$ in dimension d consists of a finite stack alphabet Γ and a finite set of transitions $T \subseteq \mathbb{N}^d \times \mathbb{N}^d \times (\Gamma \cup \bar{\Gamma} \cup \{\varepsilon\})$. Here, a configuration is a pair $(\boldsymbol{x}, s) \in \mathbb{N}^d \times \Gamma^*$. The effect of a transition $(\boldsymbol{u}, \boldsymbol{v}, \alpha)$ is subtracting $\boldsymbol{u}$ from the d counters, assuming that the counters stay non-negative, and then adding $\boldsymbol{v}$. A PVAS $\mathcal{P}$ is bidirected if $(\boldsymbol{u}, \boldsymbol{v}, a) \in T$ implies $(\boldsymbol{v}, \boldsymbol{u}, \bar{a}) \in T$. A bidirected PVASS in dimension d can be simulated by a bidirected PVAS in dimension $d + 2$ where the two additional counters add up to the number of states and specify the current state. Hence, one can reduce the reachability problem for bidirected PVASS to the reachability problem for bidirected PVAS: Given a bidirected PVAS $\mathcal{P}$ and two vectors $\boldsymbol{s}, \boldsymbol{t} \in \mathbb{N}^d$, does $(\boldsymbol{s}, \varepsilon) \xrightarrow{*} (\boldsymbol{t}, \varepsilon)$ hold?

3 Decidability

In this section, we present a simple and self-contained proof that reachability is decidable in bidirected PVASS. Consider the reachability relation between configurations with empty stack. For any states p, q, define the set $R_{p,q} \subseteq \mathbb{N}^d \times \mathbb{N}^d$ with

$$R_{p,q} = \{(\boldsymbol{u}, \boldsymbol{v}) \mid \boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^d,\ (p, \boldsymbol{u}, \varepsilon) \xrightarrow{*} (q, \boldsymbol{v}, \varepsilon)\}.$$

We will prove that each $R_{p,q}$ is semilinear, for which we rely on the fact that these sets are slices. A *slice* is a subset $S \subseteq \mathbb{N}^k$ such that if $\boldsymbol{u}, \boldsymbol{u}+\boldsymbol{v}, \boldsymbol{u}+\boldsymbol{w} \in S$ for some $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{N}^k$, then $\boldsymbol{u}+\boldsymbol{v}+\boldsymbol{w} \in S$. Observe that each $R_{p,q} \subseteq \mathbb{N}^{2d}$ is a slice. This is because if $(\boldsymbol{u}, \boldsymbol{v}), (\boldsymbol{u}+\boldsymbol{u}_1, \boldsymbol{v}+\boldsymbol{v}_1), (\boldsymbol{u}+\boldsymbol{u}_2, \boldsymbol{v}+\boldsymbol{v}_2) \in R_{p,q}$, then there is a run

$$(p, \boldsymbol{u}+\boldsymbol{u}_1+\boldsymbol{u}_2, \varepsilon) \xrightarrow{*} (q, \boldsymbol{v}+\boldsymbol{v}_1+\boldsymbol{u}_2, \varepsilon) \xrightarrow{*} (p, \boldsymbol{u}+\boldsymbol{v}_1+\boldsymbol{u}_2, \varepsilon) \xrightarrow{*} (q, \boldsymbol{v}+\boldsymbol{v}_1+\boldsymbol{v}_2, \varepsilon),$$

where the middle part exists due to bidirectedness. Thus, the pair $(\boldsymbol{u}+\boldsymbol{u}_1+\boldsymbol{u}_2, \boldsymbol{v}+\boldsymbol{v}_1+\boldsymbol{v}_2)$ belongs to $R_{p,q}$. The following was first shown in [14, Proposition 7.3].

▶ **Theorem 3.1** (Eilenberg & Schützenberger 1969). *Every slice is semilinear.*

This seems to be stronger than the somewhat better-known fact that each congruence on $\mathbb{N}^d$ is semilinear: Observe that every congruence on $\mathbb{N}^d$, seen as a subset of $\mathbb{N}^{2d}$, is a slice. In the case of congruences, a relatively simple proof was obtained by Hirshfeld [27]. We present a proof of Theorem 3.1 that combines ideas from both [14] and [27] and is (in our opinion) simpler than each.

For a set $X \subseteq \mathbb{N}^k$, let $\min X$ be the set of minimal elements of X, with respect to the usual component-wise ordering $\leq$ on $\mathbb{N}^k$. Since this ordering is a well-quasi ordering, $\min X$ is finite for every set X. Suppose $S \subseteq \mathbb{N}^k$ is a slice. For each $\boldsymbol{u} \in S$, let $S - \boldsymbol{u} := \{\boldsymbol{v} \in \mathbb{N}^k \mid \boldsymbol{u}+\boldsymbol{v} \in S\}$. Then $\boldsymbol{u} \leq \boldsymbol{v}$ implies $S - \boldsymbol{u} \subseteq S - \boldsymbol{v}$. Consider for each $\boldsymbol{u} \in S$ the submonoid

$$M_{\boldsymbol{u}} = (\min(S - \boldsymbol{u} \setminus \{\boldsymbol{0}\}))^*.$$

In other words, $M_{\boldsymbol{u}}$ is the submonoid of $\mathbb{N}^k$ generated by the non-zero minimal elements of $S - \boldsymbol{u}$. Note that for $\boldsymbol{u}, \boldsymbol{v} \in S$, we have $M_{\boldsymbol{u}} = M_{\boldsymbol{v}}$ if and only if $(S - \boldsymbol{u} \setminus \{\boldsymbol{0}\})\uparrow = (S - \boldsymbol{v} \setminus \{\boldsymbol{0}\})\uparrow$. Since S is a slice, we have $\boldsymbol{u} + M_{\boldsymbol{u}} \subseteq S$ for every $\boldsymbol{u} \in S$. Since $\boldsymbol{u} \in \boldsymbol{u} + M_{\boldsymbol{u}}$, we trivially have

$$S = \bigcup_{\boldsymbol{u} \in S} \boldsymbol{u} + M_{\boldsymbol{u}}.$$

Since each $\boldsymbol{u} + M_{\boldsymbol{u}}$ is semilinear, it suffices to show that S is covered by finitely many sets $\boldsymbol{u} + M_{\boldsymbol{u}}$. We first observe that if $\boldsymbol{u} \leq \boldsymbol{v}$ and $M_{\boldsymbol{u}} = M_{\boldsymbol{v}}$, then $\boldsymbol{u} + M_{\boldsymbol{u}}$ already covers $\boldsymbol{v} + M_{\boldsymbol{v}}$.

▶ **Lemma 3.2.** *Let $\boldsymbol{u}, \boldsymbol{v} \in S$. If $\boldsymbol{u} \leq \boldsymbol{v}$ and $M_{\boldsymbol{u}} = M_{\boldsymbol{v}}$, then $\boldsymbol{v} + M_{\boldsymbol{v}} \subseteq \boldsymbol{u} + M_{\boldsymbol{u}}$.*

Proof. We will use the following claim: For every $\boldsymbol{w} \in S$ with $\boldsymbol{u} \leq \boldsymbol{w} \leq \boldsymbol{v}$, we have $M_{\boldsymbol{u}} = M_{\boldsymbol{w}} = M_{\boldsymbol{v}}$. Indeed, since $M_{\boldsymbol{u}} = M_{\boldsymbol{v}}$, we have $\min(S - \boldsymbol{u} \setminus \{\boldsymbol{0}\}) = \min(S - \boldsymbol{v} \setminus \{\boldsymbol{0}\})$. Moreover, since S is a slice, we have $S - \boldsymbol{u} \subseteq S - \boldsymbol{w} \subseteq S - \boldsymbol{v}$. Therefore, $\min(S - \boldsymbol{w} \setminus \{\boldsymbol{0}\})$ coincides with $\min(S - \boldsymbol{u} \setminus \{\boldsymbol{0}\})$ and $\min(S - \boldsymbol{v} \setminus \{\boldsymbol{0}\})$, which implies $M_{\boldsymbol{w}} = M_{\boldsymbol{u}} = M_{\boldsymbol{v}}$.

Let us prove the lemma. We proceed by induction on $\|\boldsymbol{v} - \boldsymbol{u}\|$. If $\boldsymbol{u} = \boldsymbol{v}$, then we are done. Otherwise, there exists an $\boldsymbol{m} \in \min(S - \boldsymbol{u} \setminus \{\boldsymbol{0}\})$ such that $\boldsymbol{u} + \boldsymbol{m} \leq \boldsymbol{v}$. By our claim, we have $M_{\boldsymbol{u}} = M_{\boldsymbol{u}+\boldsymbol{m}} = M_{\boldsymbol{v}}$. Therefore, induction implies $\boldsymbol{v} \in \boldsymbol{u} + \boldsymbol{m} + M_{\boldsymbol{u}+\boldsymbol{m}}$. But since $\boldsymbol{m} \in M_{\boldsymbol{u}}$ and $M_{\boldsymbol{u}+\boldsymbol{m}} \subseteq M_{\boldsymbol{u}}$, this implies $\boldsymbol{v} \in \boldsymbol{u} + M_{\boldsymbol{u}}$. ◀

The following implies semilinearity of S.

▶ **Lemma 3.3.** *There is a finite set $F \subseteq S$ such that $S = \bigcup_{\boldsymbol{u} \in F} \boldsymbol{u} + M_{\boldsymbol{u}}$.*

Proof. Suppose not. Then there is an infinite sequence $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots \in S$ such that each set $\boldsymbol{u}_i + M_{\boldsymbol{u}_i}$ contributes a new element. By Dickson's lemma $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots$ contains a subsequence $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots$ with $\boldsymbol{v}_i \leq \boldsymbol{v}_{i+1}$ for all $i \geq 1$. Since then $S - \boldsymbol{v}_1 \subseteq S - \boldsymbol{v}_2 \subseteq \cdots$, the sequence $(S - \boldsymbol{v}_1 \setminus \{\boldsymbol{0}\})\!\uparrow \subseteq (S - \boldsymbol{v}_2 \setminus \{\boldsymbol{0}\})\!\uparrow \subseteq \cdots$ becomes stationary, again by Dickson's lemma, and therefore also the sequence $M_{\boldsymbol{v}_1}, M_{\boldsymbol{v}_2}, \ldots$. By Lemma 3.2, this means that only finitely many terms in the sequence $\boldsymbol{v}_1 + M_{\boldsymbol{v}_1}, \boldsymbol{v}_2 + M_{\boldsymbol{v}_2}, \ldots$ contribute new elements, a contradiction. ◀

Saturation invariants. We have seen that the reachability relations $R_{p,q}$ are all semilinear. However, since the semilinearity proof is non-constructive, this does not explain how to decide reachability. Nevertheless, we shall use semilinearity to show that in case of non-reachability, there exists a certificate. This yields a decision procedure consisting of two semi-algorithms in the style of Leroux's algorithm for reachability in VASS [34]: One semi-algorithm enumerates potential runs, and one enumerates potential certificates for non-reachability.

We assume that we are given a bidirected d-dimensional PVASS with state set Q and stack alphabet Γ. We may assume that all transitions are of the form $p \xrightarrow{\gamma} q$ or $p \xrightarrow{\bar{\gamma}} q$ for $\gamma \in \Gamma$ or $p \xrightarrow{\boldsymbol{v}} q$ for $\boldsymbol{v} \in \mathbb{Z}^d$. Our certificates for non-reachability will be in the form of what we call saturation invariants. Imagine a (non-terminating) naive saturation algorithm that attempts to compute the sets $R_{p,q}$ by adding vector pairs one-by-one to finite sets $F_{p,q}$. It would start with $F_{p,q} = \emptyset$ and then add pairs: For each transition $p \xrightarrow{\boldsymbol{v}} q$ and each vector $\boldsymbol{u} \in \mathbb{N}^d$ with $\boldsymbol{u} + \boldsymbol{v} \in \mathbb{N}^d$, it would add the pair $(\boldsymbol{u}, \boldsymbol{u} + \boldsymbol{v})$ to $F_{p,q}$. Moreover, if $(\boldsymbol{u}, \boldsymbol{v}) \in F_{p,q}$ and $(\boldsymbol{v}, \boldsymbol{w}) \in F_{q,r}$, it would add $(\boldsymbol{u}, \boldsymbol{w})$ to $F_{p,r}$. Finally, if there are transitions $p \xrightarrow{\gamma} p'$ and $q' \xrightarrow{\bar{\gamma}} q$ and there is a $(\boldsymbol{u}, \boldsymbol{v}) \in F_{p',q'}$, then it would add $(\boldsymbol{u}, \boldsymbol{v})$ to $F_{p,q}$.

Intuitively, a saturation invariant is a forward inductive invariant of this naive saturation algorithm. Let us make this precise. For subsets $R_1, R_2 \subseteq \mathbb{N}^d \times \mathbb{N}^d$, we define

$$R_1 \circ R_2 = \{(\boldsymbol{u}, \boldsymbol{w}) \in \mathbb{N}^d \times \mathbb{N}^d \mid \exists \boldsymbol{v} \in \mathbb{N}^d \colon (\boldsymbol{u}, \boldsymbol{v}) \in R_1,\ (\boldsymbol{v}, \boldsymbol{w}) \in R_2\}.$$

A *saturation invariant* consists of a family $(I_{p,q})_{(p,q) \in Q^2}$ of sets $I_{p,q} \subseteq \mathbb{N}^d \times \mathbb{N}^d$ for which

1. For each transition $p \xrightarrow{\boldsymbol{v}} q$, $\boldsymbol{v} \in \mathbb{Z}^d$, each $\boldsymbol{u} \in \mathbb{N}^d$ with $\boldsymbol{u}+\boldsymbol{v} \in \mathbb{N}^d$, we have $(\boldsymbol{u}, \boldsymbol{u}+\boldsymbol{v}) \in I_{p,q}$.
2. For each $p, q, r \in Q$, we have $I_{p,q} \circ I_{q,r} \subseteq I_{p,r}$.
3. For each $p, p', q, q' \in Q$ for which there are transitions $p \xrightarrow{\gamma} p'$, $q' \xrightarrow{\bar{\gamma}} q$ for some $\gamma \in \Gamma$, we have $I_{p',q'} \subseteq I_{p,q}$.

There is a natural ordering of such families $(I_{p,q})_{(p,q) \in Q^2}$ defined by inclusion: We write $(I_{p,q})_{(p,q) \in Q^2} \leq (J_{p,q})_{(p,q) \in Q^2}$, if $I_{p,q} \subseteq J_{p,q}$ for each $p, q \in Q$. In this sense, we can speak of a smallest saturation invariant.

▶ **Lemma 3.4.** *The family $(R_{p,q})_{(p,q) \in Q^2}$ is the smallest saturation invariant.*

Proof. By induction on the length of a run, it follows that $(R_{p,q})_{(p,q) \in Q^2}$ is included in every saturation invariant. Moreover, $(R_{p,q})_{(p,q) \in Q^2}$ is clearly a saturation invariant itself. ◀

Our certificates will consist of saturation invariants defined in Presburger arithmetic. A family $(I_{p,q})_{(p,q) \in Q^2}$ is *Presburger-definable* if for each $(p, q) \in Q^2$, the set $I_{p,q}$ is semilinear. According to Theorem 3.1, the family $(R_{p,q})_{(p,q) \in Q^2}$ is Presburger-definable. Therefore, the following is a direct consequence of Lemma 3.4.

▶ **Theorem 3.5.** *For each $s, t \in Q$, we have $(\boldsymbol{0}, \boldsymbol{0}) \notin R_{s,t}$ if and only if there exists a Presburger-definable saturation invariant $(I_{p,q})_{(p,q) \in Q^2}$ such that $(\boldsymbol{0}, \boldsymbol{0}) \notin I_{s,t}$.*

Algorithm 1 Algorithm for bidirected reachability in PVAS.

Data: Bidirected d-dim. PVAS $\mathcal{P} = (\Gamma, T)$

$R_0 := \mathsf{Cong}(\{(\boldsymbol{u}, \boldsymbol{v}) \mid (\boldsymbol{u}, \boldsymbol{v}, \varepsilon) \in T\})$;
for $i = 1, 2, \ldots$ **do**
  $R_i \leftarrow R_{i-1}$;
  for $(\boldsymbol{u}, \boldsymbol{u}', a) \in T$ *and* $(\boldsymbol{v}', \boldsymbol{v}, \bar{a}) \in T$ **do**
    $R_i \leftarrow R_i \cup \{(\boldsymbol{x} + \boldsymbol{u}, \boldsymbol{y} + \boldsymbol{v}) \mid (\boldsymbol{x} + \boldsymbol{u}', \boldsymbol{y} + \boldsymbol{v}') \in R_{i-1}, \, \boldsymbol{x}, \boldsymbol{y} \in \mathbb{N}^d\}$;
  $R_i \leftarrow \mathsf{Cong}(R_i)$;
  if $R_i = R_{i-1}$ **then return** R_i;

This yields our algorithm: One semi-algorithm enumerates transition sequences and terminates if one of them is a run witnessing $(s, \mathbf{0}, \varepsilon) \xrightarrow{*} (t, \mathbf{0}, \varepsilon)$. The other semi-algorithm enumerates Presburger-definable families $(I_{p,q})_{(p,q) \in Q^2}$ in the form of Presburger formulas. Using Presburger arithmetic, it is then easy to check whether (i) $(I_{p,q})_{(p,q) \in Q^2}$ is a saturation invariant and (ii) $(\mathbf{0}, \mathbf{0}) \notin I_{s,t}$. If a saturation invariant is found, the semi-algorithm reports non-reachability. By Theorem 3.5, one of the two semi-algorithms must terminate.

4 Ackermann upper bound

In this section, we show that reachability in bidirected PVASS is solvable in Ackermann time in the general case and in primitive recursive complexity in every fixed dimension.

One way to avoid enumeration in the algorithm of Section 3 would be to start with the semilinear one-step relation described in the first condition of saturation invariants, and then to enlarge it according to the second and third condition. Moreover, one could take the slice closure (the smallest slice that includes the current set) after each enlargement. Since slices satisfy an ascending chain condition [14, Corollary 12.3], this would ensure termination. In fact, computing the slice closure of a semilinear set is possible with an algorithm by Grabowski [22]. Unfortunately, the latter is itself based on enumeration and we are not aware of any complexity bounds for computing slice closures. Therefore, we use an analogous algorithm that uses congruences instead of slices. Since congruences can be encoded in polynomial ideals, we can tap into the rich toolbox of Gröbner bases to compute the congruence generated by a semilinear set.

4.1 The saturation algorithm

In the following we will work with pushdown VAS instead of pushdown VASS. Our decision procedure for bidirected reachability relies on the crucial fact that the reachability relation $R_{\mathcal{P}} = \{(\boldsymbol{s}, \boldsymbol{t}) \in \mathbb{N}^d \times \mathbb{N}^d \mid (\boldsymbol{s}, \varepsilon) \xrightarrow{*} (\boldsymbol{t}, \varepsilon)\}$ of a bidirected pushdown VAS $\mathcal{P}$ is a congruence: It is always reflexive, transitive and additive, even for directed pushdown VAS, and symmetric for bidirected systems. Therefore, whenever we have found an underapproximation $R \subseteq R_{\mathcal{P}}$ we can replace R by the smallest congruence containing R. The smallest congruence containing a set $R \subseteq \mathbb{N}^d \times \mathbb{N}^d$ is denoted by $\mathsf{Cong}(R)$. We also say that R is a *basis of* (or *generates*) $\mathsf{Cong}(R)$. Recall that every congruence on $\mathbb{N}^d$ is a slice. Therefore, congruences are semilinear and ascending chains of congruences stabilize.

Algorithm 1 is a saturation algorithm that computes a semilinear representation for $R_{\mathcal{P}}$. The sets R_i are maintained by semilinear representations or Presburger formulas. Since in this section we only prove elementary complexity bounds, we can use both formats interchangeably. Observe that the update in line 5 and the equality test in line 7 can be expressed in Presburger arithmetic. The computation of $\mathsf{Cong}(\cdot)$ will be explained in the next subsection. Consider the values of R_i for $i \geq 1$ after line 6 of Algorithm 1. They form an ascending chain of congruences $(R_i)_{i\geq 1}$, which implies that the algorithm must terminate. For the correctness one can prove by induction on i that $(\boldsymbol{x}, \boldsymbol{y}) \in R_i$ if and only if there exists a run between $(\boldsymbol{x}, \varepsilon)$ and $(\boldsymbol{y}, \varepsilon)$ whose stack height does not exceed i. Moreover, if the algorithm terminates after k iterations then $R_k = R_{\mathcal{P}}$.

We will use a primitive recursive algorithm (which is elementary in fixed dimension) to compute $\mathsf{Cong}(R_i)$ from a semilinear representation for R_i. Using the tools from [49] we can then prove upper bounds for the length of the ascending chain.

4.2 Semilinear representations for congruences

In this section, we present an algorithm that, for a given semilinear representation for a set $R \subseteq \mathbb{N}^d \times \mathbb{N}^d$, computes a semilinear representation for $\mathsf{Cong}(R)$. Its run time is bounded by a tower of exponentials in $\|R\|$ of height $O(d)$ (Theorem 4.4). Note that for bidirected VASS, it is known that in exponential space, one can compute a semilinear representation of the reachability set [32, 6]. In other words, one can compute in exponential space a representation of the congruence class of a given vector $\boldsymbol{x} \in \mathbb{N}^d$. In contrast, our algorithm computes a semilinear representation of the entire congruence.

Let the function $\exp^k$ be inductively defined by $\exp^0(x) = x$ and $\exp^{k+1}(x) = \exp^k(2^x)$. In the following we show how to compute a semilinear representation for a congruence $\mathcal{Q}$ given by a semilinear basis $R \subseteq \mathbb{N}^d \times \mathbb{N}^d$ in time $\exp^{O(d)}(\|R\|)$. In fact, we can assume that R is *finite* since a linear set $L = \boldsymbol{b} + P^*$ is contained in $\mathsf{Cong}(F_L)$ where $F_L = \{\boldsymbol{b}, \boldsymbol{b}+\boldsymbol{p} \mid \boldsymbol{p} \in P\}$, and, therefore, a semilinear set $\bigcup_{i=1}^m L_i$ generates the same congruence as $\bigcup_{i=1}^m F_{L_i}$. The semilinear representation of $\mathcal{Q}$ will be obtained by induction on the dimension d via a decomposition of $\mathbb{N}^d$ into smaller regions. A *region* is a linear set $L = \boldsymbol{b} + P^* \subseteq \mathbb{N}^d$ where $P \subseteq \{\boldsymbol{e}_1, \dots, \boldsymbol{e}_d\}$. Its *dimension* is $|P|$. In particular all sets $\boldsymbol{b}{\uparrow} = \boldsymbol{b} + \{\boldsymbol{e}_1, \dots, \boldsymbol{e}_d\}^*$ are regions. For a region $L = \boldsymbol{b} + P^*$ and a congruence $\mathcal{Q}$, we define the congruence $\mathcal{Q}_L = \{(\boldsymbol{x}, \boldsymbol{y}) \in (P^*)^2 \mid (\boldsymbol{b}+\boldsymbol{x}, \boldsymbol{b}+\boldsymbol{y}) \in \mathcal{Q}\}$ on the submonoid P^*.

A submonoid $S \subseteq \mathbb{N}^k$ is *subtractive* if $\boldsymbol{x}, \boldsymbol{y} \in S$ and $\boldsymbol{x} \leq \boldsymbol{y}$ implies $\boldsymbol{y} - \boldsymbol{x} \in S$. For example, the non-negative restriction $G \cap \mathbb{N}^k$ of a group $G \subseteq \mathbb{Z}^k$ is a subtractive submonoid. The following lemma is well-known, see [14, Proposition 7.1] or the full version [20] for a proof.

▶ **Lemma 4.1.** *Every subtractive submonoid $S \subseteq \mathbb{N}^k$ is of the form $S = M^*$ where M is the finite set of the minimal nonzero elements in S.*

Eilenberg and Schützenberger observed that for every slice S there exists an element $\boldsymbol{s} \in S$ such that $S - \boldsymbol{s}$ is a subtractive submonoid [14, Proposition 7.2]. As a consequence, for every congruence $\mathcal{Q}$ on $\mathbb{N}^d$ there exists $\boldsymbol{b} \in \mathbb{N}^d$ such that $\mathcal{Q}_{\boldsymbol{b}\uparrow}$ is a subtractive submonoid. We provide an elementary bound on $\boldsymbol{b}$.

▶ **Lemma 4.2.** *Given a finite basis R for a congruence $\mathcal{Q}$ on $\mathbb{N}^d$, one can compute in elementary time a vector $\boldsymbol{b} \in \mathbb{N}^d$ and a finite set $M \subseteq \mathbb{N}^{2d}$ such that $\mathcal{Q}_{\boldsymbol{b}\uparrow} = M^*$.*

Gröbner bases. It remains to compute a semilinear representation of $\mathcal{Q}$ on the complement of $\boldsymbol{b}{\uparrow}$. We will decompose $\mathbb{N}^d \setminus \boldsymbol{b}{\uparrow}$ into disjoint $(d-1)$-dimensional regions L_j, compute bases for the restrictions $\mathcal{Q}_{L_j}$, and proceed inductively. To compute the bases for $\mathcal{Q}_{L_j}$, we will exploit the well-studied connection between congruences on $\mathbb{N}^d$ and binomial ideals [43]. Let $\mathbb{Z}[\boldsymbol{x}]$ be the polynomial ring in the variables $\boldsymbol{x} = (x_1, \dots, x_d)$ over $\mathbb{Z}$. We write $\boldsymbol{x}^{\boldsymbol{u}}$ for the monomial $x_1^{\boldsymbol{u}(1)} \cdots x_d^{\boldsymbol{u}(d)}$. An *ideal* is a nonempty set $I \subseteq \mathbb{Z}[\boldsymbol{x}]$ such that $f, g \in I$ and $h \in \mathbb{Z}[\boldsymbol{x}]$ implies $f + g, hf \in I$. An ideal I is finitely represented by a *basis* B, i.e. I is the smallest ideal containing B. By Hilbert's basis theorem any ideal $I \subseteq \mathbb{Z}[\boldsymbol{x}]$ has a finite basis. One of the main tools in computer algebra for handling polynomial ideals are *Gröbner bases*, e.g. to solve the ideal membership problem. We defer the reader to [2] for details on Gröbner bases and only mention the properties required for our purposes. A Gröbner basis is defined with respect to an admissible monomial ordering, e.g. a lexicographic ordering on the monomials. Buchberger's algorithm [7] computes for a given basis for an ideal I the *unique reduced* Gröbner basis for I in doubly exponential space [13].

A basis R for a congruence $\mathcal{Q}$ on $\mathbb{N}^d$ can be translated into the polynomial ideal $I \subseteq \mathbb{Z}[\boldsymbol{x}]$ generated by $B_R = \{\boldsymbol{x}^{\boldsymbol{u}} - \boldsymbol{x}^{\boldsymbol{v}} \mid (\boldsymbol{u}, \boldsymbol{v}) \in R\}$. It is known that $\boldsymbol{s} \sim_{\mathcal{Q}} \boldsymbol{t}$ if and only if $\boldsymbol{x}^{\boldsymbol{s}} - \boldsymbol{x}^{\boldsymbol{t}} \in I$ [43, Lemma 1 and 2]. Moreover, the reduced Gröbner basis of I with respect to an admissible monomial order always consists of differences of monomials $\boldsymbol{x}^{\boldsymbol{s}} - \boldsymbol{x}^{\boldsymbol{t}}$ [29, Theorem 2.7].

The following lemma can be reduced to two known applications of Gröbner bases. Let $I \subseteq \mathbb{Z}[\boldsymbol{x}]$ be an ideal. For a subsequence $\boldsymbol{y}$ of $\boldsymbol{x}$ we call $I \cap \mathbb{Z}[\boldsymbol{y}]$ the *elimination ideal*, which is indeed an ideal in $\mathbb{Z}[\boldsymbol{y}]$. For $\boldsymbol{b} \in \mathbb{N}^d$ we define the *ideal quotient* $I : \boldsymbol{x}^{\boldsymbol{b}} = \{p \in \mathbb{Z}[\boldsymbol{x}] \mid p\boldsymbol{x}^{\boldsymbol{b}} \in I\}$, which is also an ideal. It is known that one can compute Gröbner bases for $I \cap \mathbb{Z}[\boldsymbol{y}]$ and $I : \boldsymbol{x}^{\boldsymbol{b}}$ in elementary time [2, Section 6], see the full version [20] for more details.

▶ **Lemma 4.3.** *Given a finite basis R for a congruence $\mathcal{Q}$ on $\mathbb{N}^d$ and a region $L \subseteq \mathbb{N}^d$, one can compute in elementary time a finite basis for $\mathcal{Q}_L$.*

We are ready to compute a semilinear representation of $\mathsf{Cong}(R)$ in $\exp^{O(d)}(n)$ time. We proceed by induction over d. Using Lemma 4.2 we can write $\mathcal{Q}_{\boldsymbol{b}\uparrow} = M^*$. We decompose $\mathbb{N}^d = \bigcup_{i=0}^m L_j$ into regions where $L_0 = \boldsymbol{b}\uparrow$ and $L_1, \dots, L_m$ are $(d-1)$-dimensional regions. By Lemma 4.3 we can compute bases for $\mathcal{Q}_{L_i}$ for $i \in [1, m]$ and by induction hypothesis semilinear representations for $\mathcal{Q}_{L_i}$. In this way, we obtain semilinear representations for the restrictions $Q_i = \mathcal{Q} \cap L_i^2$ for each $i \in [0, m]$. Finally, we can express $\boldsymbol{s} \sim_{\mathcal{Q}} \boldsymbol{t}$ by a Presburger formula that says that there exists a sequence of intermediate vectors of length $2(m+1)$ where adjacent elements are related by an R-step or are contained in some relation Q_i.

▶ **Theorem 4.4.** *Given a semilinear basis R for a congruence $\mathcal{Q}$ on $\mathbb{N}^d$, one can compute a semilinear representation for $\mathcal{Q}$ in time $\exp^{c_1 d}(n)$ for some absolute constant c_1.*

4.3 Ascending chains of congruences

To bound the length of the chain of congruences R_i in Algorithm 1 we use a *length function theorem* [49, Theorem 3.15], see also [16]. In general, such theorems allow to derive complexity bounds for algorithms whose termination arguments are based on well-quasi orders.

Fast-growing complexity classes. In the following we state a simplified version of [49, Theorem 3.15], which is sufficient for our application. We start by introducing fast-growing functions and complexity classes. Recall that the Cantor normal form of an ordinal $\alpha \le \omega^\omega$ is the unique representation $\alpha = \omega^{\alpha_1} + \cdots + \omega^{\alpha_p}$ where $\alpha > \alpha_1 \ge \cdots \ge \alpha_p$. In this form α is a limit ordinal if and only if $p > 0$ and $\alpha_p > 0$. A *fundamental sequence* for a limit ordinal λ is a sequence $(\lambda(x))_{x<\omega}$ of ordinals with supremum λ. Given a limit ordinal $\lambda \le \omega^\omega$ whose Cantor normal form is $\lambda = \beta + \omega^{k+1}$, we use the standard fundamental sequence $(\lambda(x))_{x<\omega}$, defined inductively as $\omega^\omega(x) = \omega^{x+1}$ and $(\beta + \omega^{k+1})(x) = \beta + \omega^k \cdot (x+1)$. Given a function $h \colon \mathbb{N} \to \mathbb{N}$ the *Hardy hierarchy* $(h^\alpha)_{\alpha \le \omega^\omega}$ relative to h is defined by

$$h^0(x) = x, \qquad h^{\alpha+1}(x) = h^\alpha(h(x)), \qquad h^\lambda(x) = h^{\lambda(x)}(x).$$

Using the Hardy functions $(H^\alpha)_\alpha$ relative to $H(x) = x + 1$ we can define the *fast-growing* complexity classes $(\mathsf{F}_\alpha)_\alpha$ from [48]. We denote by $\mathscr{F}_{<\alpha}$ the class of functions computed by deterministic Turing machines in time $O(H^\beta(n))$ for some $\beta < \omega^\alpha$. By F_α we denote the class of decision problems solved by deterministic Turing machines in time $O(H^{\omega^\alpha}(p(n)))$ for some function $p \in \mathscr{F}_{<\alpha}$. We define $\mathsf{PRIMITIVE\text{-}RECURSIVE} = \bigcup_{k<\omega} \mathsf{F}_k$ and $\mathsf{ACKERMANN} = \mathsf{F}_\omega$.

Controlled bad sequences. By Dickson's lemma, any sequence of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ with $x_i \not\leq x_j$ for all $i < j$ must be finite. Such a sequence is also called *bad*. To obtain complexity bounds on the length of bad sequences we need to restrict to sequences that do not grow in an uncontrolled fashion. In the following let $g\colon \mathbb{N} \to \mathbb{N}$ be *monotone*, *strictly inflationary*, i.e. $g(x) > x$ for all x, and *super-homogeneous*, i.e. $g(xy) \geq g(x) \cdot y$ for all $x, y \geq 1$. A sequence of vectors $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$ is (g, n)-*controlled* if $\|\boldsymbol{x}_i\| \leq g^i(n)$ for all $i \in [0, \ell]$. The following statement follows from [49, Theorem 3.15] and [49, Eq. (3.13)].

▶ **Theorem 4.5.** *Any (g, n)-controlled bad sequence over $\mathbb{N}^k$ has length at most $g^{\omega^k}(nk)$.*

A *chain* in $\mathbb{N}^k$ is a sequence $S_0 \subsetneq S_1 \subsetneq \cdots \subsetneq S_\ell$ of subsets $S_i \subseteq \mathbb{N}^k$. The chain is (g, n)-*controlled* if for each $i \in [0, \ell - 1]$ there exists $\boldsymbol{s}_i \in S_{i+1} \setminus S_i$ with $\|\boldsymbol{s}_i\| \leq g^i(n)$. A set $X \subseteq \mathbb{N}^k$ is *upwards closed* if $X = X\!\uparrow$. Observe that any (g, n)-controlled chain of upwards closed sets in $\mathbb{N}^k$ is bounded by $1 + g^{\omega^k}(nk)$ since we obtain a (g, n)-controlled bad sequence $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{\ell-1}$ by picking arbitrary $\boldsymbol{s}_i \in S_{i+1} \setminus S_i$ with $\|\boldsymbol{s}_i\| \leq g^i(n)$.

Translating congruences into upwards closed sets. The key trick in our upper bound for ascending chains of congruences is to translate congruences into upwards closed sets. This allows us to translate bounds on the length of ascending chains of upward closed sets into corresponding bounds for congruences. The translation works as follows. To each congruence $\mathcal{Q}$ on $\mathbb{N}^d$, we associate the upwards closed set $U(\mathcal{Q}) \subseteq \mathbb{N}^{4d}$ with

$$U(\mathcal{Q}) = \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v}) \mid (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Q},\ (\boldsymbol{x} + \boldsymbol{u}, \boldsymbol{y} + \boldsymbol{v}) \in \mathcal{Q},\ (\boldsymbol{u}, \boldsymbol{v}) \neq (\boldsymbol{0}, \boldsymbol{0})\}\!\uparrow.$$

Clearly $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$ implies $U(\mathcal{Q}_1) \subseteq U(\mathcal{Q}_2)$. In fact, strict inclusion is also preserved:

▶ **Lemma 4.6.** *Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be congruences with $\mathcal{Q}_1 \subsetneq \mathcal{Q}_2$. Then (i) $U(\mathcal{Q}_1) \subsetneq U(\mathcal{Q}_2)$ and (ii) for each $\boldsymbol{q} \in \mathcal{Q}_2 \setminus \mathcal{Q}_1$, there is a $\boldsymbol{p} \in U(\mathcal{Q}_2) \setminus U(\mathcal{Q}_1)$ with $\|\boldsymbol{p}\| \leq \|\boldsymbol{q}\|$.*

Proof. Statement (i) is immediate. For statement (ii) let $(\boldsymbol{s}, \boldsymbol{t}) \in \mathcal{Q}_2 \setminus \mathcal{Q}_1$ be minimal. Since $(\boldsymbol{0}, \boldsymbol{0}) \in \mathcal{Q}_1$ we must have $(\boldsymbol{s}, \boldsymbol{t}) \neq (\boldsymbol{0}, \boldsymbol{0})$ and there exists $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Q}_1$ with $(\boldsymbol{x}, \boldsymbol{y}) < (\boldsymbol{s}, \boldsymbol{t})$. We choose such a vector $(\boldsymbol{x}, \boldsymbol{y})$ in which $(\boldsymbol{u}, \boldsymbol{v}) := (\boldsymbol{s}, \boldsymbol{t}) - (\boldsymbol{x}, \boldsymbol{y})$ is minimal. Clearly, $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v}) \in U(\mathcal{Q}_2)$. We claim that $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v}) \notin U(\mathcal{Q}_1)$. Towards a contradiction, suppose that there exists $(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{u}_1, \boldsymbol{v}_1) \leq (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v})$ with $(\boldsymbol{x}_1, \boldsymbol{y}_1) \in \mathcal{Q}_1$, $(\boldsymbol{u}_1, \boldsymbol{v}_1) \neq (\boldsymbol{0}, \boldsymbol{0})$, and $(\boldsymbol{x}_1 + \boldsymbol{u}_1, \boldsymbol{y}_1 + \boldsymbol{v}_1) \in \mathcal{Q}_1$. Since $\mathcal{Q}_1$ is a congruence we have

$$\begin{aligned}\boldsymbol{x} + \boldsymbol{u}_1 = (\boldsymbol{x} - \boldsymbol{x}_1) + \boldsymbol{x}_1 + \boldsymbol{u}_1 &\sim_{\mathcal{Q}_1} (\boldsymbol{x} - \boldsymbol{x}_1) + \boldsymbol{y}_1 + \boldsymbol{v}_1 \\ &\sim_{\mathcal{Q}_1} (\boldsymbol{x} - \boldsymbol{x}_1) + \boldsymbol{x}_1 + \boldsymbol{v}_1 = \boldsymbol{x} + \boldsymbol{v}_1 \sim_{\mathcal{Q}_1} \boldsymbol{y} + \boldsymbol{v}_1.\end{aligned}$$

If $(\boldsymbol{u}_1, \boldsymbol{v}_1) = (\boldsymbol{u}, \boldsymbol{v})$ then this contradicts $(\boldsymbol{x} + \boldsymbol{u}, \boldsymbol{y} + \boldsymbol{v}) = (\boldsymbol{s}, \boldsymbol{t}) \notin \mathcal{Q}_1$. If $(\boldsymbol{u}_1, \boldsymbol{v}_1) < (\boldsymbol{u}, \boldsymbol{v})$ then $(\boldsymbol{s}, \boldsymbol{t}) = (\boldsymbol{x} + \boldsymbol{u}_1, \boldsymbol{y} + \boldsymbol{v}_1) + (\boldsymbol{u} - \boldsymbol{u}_1, \boldsymbol{v} - \boldsymbol{v}_1)$ contradicts the minimality of $(\boldsymbol{u}, \boldsymbol{v})$. ◀

If $(\mathcal{Q}_i)_{i \leq \ell}$ is a (g, n)-controlled chain of congruences in $\mathbb{N}^d$ then by Lemma 4.6, $(U(\mathcal{Q}_i))_{i \leq \ell}$ is a (g, n)-controlled chain of upwards closed subsets of $\mathbb{N}^{4d}$ and thus has length at most $1 + g^{\omega^{4d}}(4dn)$. Hence, the same bound applies to $(\mathcal{Q}_i)_{i \leq \ell}$.

▶ **Lemma 4.7.** *Any* (g, n)*-controlled chain of congruences in* $\mathbb{N}^d$ *has length* $\leq 1 + g^{\omega^{4d}}(4dn)$*.*

Putting together Theorem 4.4 and Lemma 4.7 we can conclude that Algorithm 1 terminates after $H^{\omega^{4d+3}}(e(n))$ iterations for some elementary function $e(n)$.

▶ **Proposition 4.8.** *Reachability in bidirected pushdown VAS is in* ACKERMANN*, and in* F_{4d+3} *if the dimension* d *is fixed.*

The detailed complexity analysis can be found in the full version [20]. The result above also holds for bidirected pushdown *VASS* (Theorem 1.1) by simulating the state in two additional counters. Hence the complexity for dimension d increases from F_{4d+3} to F_{4d+11}.

5 One-dimensional pushdown VASS

In this section we prove that reachability is in polynomial space for bidirected 1-PVASS (Theorem 1.2). For the rest of this section consider a 1-PVASS $\mathcal{P} = (Q, \Gamma, T)$ of $|Q| = n$ states. To simplify our bounds, we assume $|\Gamma| = 2$. This can be achieved with a simple encoding: To simulate stack letters $a_1, \ldots, a_k$, we can encode each a_i by the string $ab^iab^{k-i}a$.

Preliminaries. We extend the usual component-wise ordering $\leq$ to tuples $(\mathbb{Z} \cup \{-\infty, \omega\})^k$. Given two functions $f, g \colon X \to (\mathbb{Z} \cup \{-\infty, \omega\})^k$, we write $f \leq g$ to denote that $f(x) \leq g(x)$ for each $x \in X$. We define the *one-step* $\mathbb{Z}$ *relation* $\hookrightarrow$ for $\mathcal{P}$ similarly to the one-step relation $\to$ but with a $\mathbb{Z}$-counter, i.e., we do not require the counter to remain non-negative. A *path from* p *to* q consists of the initial state p and a sequence of transitions of $\mathcal{P}$, such that it induces a run $(p_1, x_1, w_1) \hookrightarrow (p_2, x_2, w_2) \hookrightarrow \ldots \hookrightarrow (p_j, x_j, w_j)$, with the requirement that $p_1 = p$, $x_1 = 0$ and $w_1 = w_j = \varepsilon$. Given such a path P, we let

- $\mathit{MaxSH}(P) = \max_i |w_i|$ be the maximum stack height along P,
- $w(P) = x_j$ be the value of the counter at the end of P, and
- $m(P) = \min_i x_i$ be the minimum value of the counter along P. Note that $m(P) \leq 0$, as the counter is 0 at the beginning of P.

We also write $\overline{P}$ for the reverse of P. We denote by $\{p \rightsquigarrow q\}$ the set of paths from p to q, and let $\{p \rightsquigarrow q\}_k = \{P \in \{p \rightsquigarrow q\} \colon \mathit{MaxSH}(P) \leq k\}$ be the set of such paths with stack height at most k. Given two paths P_1 and P_2, we write $P_1 \leq P_2$ to denote that $(m(P_1), w(P_1)) \leq (m(P_2), w(P_2))$.

We say that a state q is reachable from a state p if $(p, \mathbf{0}, \varepsilon) \xrightarrow{*} (q, \mathbf{0}, \varepsilon)$. We say that q is $\mathbb{Z}$*-reachable* from p if there is a path $P \in \{p \rightsquigarrow q\}$ with $w(P) = 0$ (hence state reachability implies $\mathbb{Z}$-reachability). Given additionally a natural number $i \in \mathbb{N}$, we say that p *covers* (q, i) if $(p, 0, \varepsilon) \xrightarrow{*} (q, i + j, \varepsilon)$ for some $j \geq 0$. Thus reachability implies coverability for $i = 0$. The following is a simpler proof of a reduction observed in [31]: For bidirected 1-PVASS, reachability reduces to coverability and $\mathbb{Z}$-reachability.

▶ **Lemma 5.1.** *For any two states* p, q *of* $\mathcal{P}$*, we have that* p *reaches* q *iff (i)* p *covers* $(q, 0)$*, (ii)* q *covers* $(p, 0)$*, and (iii)* p $\mathbb{Z}$*-reaches* q*.*

Proof. Clearly if p reaches q then conditions (i)-(iii) hold, so we only need to argue about the reverse direction. If p does not cover $(q, 1)$, since p covers $(q, 0)$, we have that p reaches q, and we are done. Similarly, if q does not cover $(p, 1)$, we have that q reaches p and thus we are done. Finally, assume that p covers $(q, 1)$ and q covers $(p, 1)$, and let P_p and P_q be the corresponding paths witnessing coverability. Let P be a path witnessing that p $\mathbb{Z}$-reaches q. We construct the path H_ℓ witnessing the reachability of q from p as $H_\ell = (P_p \circ P_q)^\ell \circ P \circ (\overline{P_p} \circ \overline{P_q})^\ell$, where ℓ is chosen such that $w((P_p \circ P_q)^\ell) \geq -m(P)$. ◀

In light of Lemma 5.1, for Theorem 1.2, it suffices to show that for bidirected 1-PVASS, both $\mathbb{Z}$-reachability and coverability can be decided in PSPACE. The former is known already: Reachability in $\mathbb{Z}$-PVASS belongs to NP [24]; in the bidirected case, it is even decidable in P [21]. Thus, the rest of this section is devoted to deciding coverability in PSPACE.

▶ **Lemma 5.2.** *Coverability in 1-dimensional bidirected pushdown VASS is in* PSPACE.

Summary functions. We define a summary function $\gamma_k \colon Q \times Q \to (\mathbb{Z}_{\leq 0} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{-\infty, \omega\})$, parametric on $k \in \mathbb{N}$, as $\gamma_k(p,q) = (a,b)$, where a and b are defined as follows.

$$a = \max(\{m(P) \colon P \in \{p \rightsquigarrow q\}_k\}) \qquad b = \sup(\{w(P) \colon P \in \{p \rightsquigarrow q\}_k \text{ and } m(P) = a\})$$

with the convention that $\max(\emptyset) = \sup(\emptyset) = -\infty$. We occasionally write $\gamma_k(p,q) = (a, \_)$ to denote that $\gamma_k(p,q) = (a,b)$ for some b. We further define a summary function $\delta_k \colon V \to (\mathbb{Z}_{\leq 0} \cup \{-\infty\})$, parametric on $k \in \mathbb{N}$, as follows.

$$\delta_k(p) = \max(\{m(P) \colon P \in \{p \rightsquigarrow p\}_k \text{ and } w(P) > 0\})$$

Recall that we use $n = |Q|$ for the number of states in $\mathcal{P}$. It is well-known that in any pushdown system of n states (and only two stack letters), a shortest path between two states has length $2^{O(n^2)}$ (e.g. this follows by inspecting a proof of the pumping lemma for context-free languages [28, Lemma 6.1]; a precise bound was obtained in [45]). Since both the weight and the minima of any path are lower-bounded by minus the length of the path, if $\{p \rightsquigarrow q\}_k \neq \emptyset$, then a shortest path in $\{p \rightsquigarrow q\}_k$ witnesses $\gamma_k(p,q) = (a,b)$ where a and b are at most exponentially negative. This is established in the following lemma.

▶ **Lemma 5.3.** *Consider any two states p, q and natural number k, and let $\gamma_k(p,q) = (a,b)$. If $a > -\infty$ then $a, b \geq -\beta$, for $\beta = 2^{O(n^2)}$.*

The bidirectedness of $\mathcal{P}$ also leads to the following lemma.

▶ **Lemma 5.4.** *Consider any two states p, q and natural number k, and let $\gamma_k(p,q) = (a,b)$. There exists a constant α independent of $\mathcal{P}$ and k, such that, if $b > 2^{\alpha n^2}$, then $b = \omega$.*

The intuition behind the summary functions γ_k and δ_k is as follows. Lemma 5.3 and Lemma 5.4 hint on an algorithm to decide coverability by saturating γ_k iteratively for increasing k. The lemmas state that the finite values of γ_k are exponentially bounded, and thus the process is guaranteed to reach a fixpoint within exponentially many iterations. An obstacle to this approach is that γ_k may fail to capture paths that are useful in subsequent iterations. In particular, $\gamma_k(p,q)$ misses paths that can reach q with a larger counter at the cost of a lower minima on the way. The following lemma shows that δ_k captures the effects of all paths missed by γ_k, and allows the two summaries to be *mutually* saturated.

▶ **Lemma 5.5.** *For any states p, q, let $\gamma_k(p,q) = (a,b)$, and assume there exists a path $P \in \{p \rightsquigarrow q\}_k$ with $w(P) > b$. Then $\delta_k(p) \geq m(P)$.*

Moreover, the values of δ_k are also exponentially bounded, and thus the mutual saturation can be carried out in polynomial space.

▶ **Lemma 5.6.** *For any state p, if $\delta_k(p) > -\infty$ then $\delta_k(p) \geq -\beta$, for $\beta = 2^{O(n^2)}$.*

In the remainder of this section we describe the saturation procedure for γ_k and δ_k.

Finite graphs. We consider weighted finite graphs $G = (V_G, E_G, w_G)$ where $w_G\colon E_G \to \mathbb{Z}$. Moreover, we assume that every connected component of G is strongly connected. By a small abuse we extend some of the above notation on $\mathcal{P}$ to such graphs. Given two nodes u, v in G, we write $\{u \rightsquigarrow v\}^G$ for the set of paths from u to v in G. We similarly extend the summary functions to γ^G and δ^G, defined by the corresponding paths $P \in \{u \rightsquigarrow v\}^G$.

▶ **Lemma 5.7.** *Given a graph G as above, the summary functions γ^G and δ^G can be computed in polynomial time.*

Computing γ_k and δ_k. We now describe a dynamic-programming algorithm for computing γ_k and δ_k, for increasing values of k. We let $\Gamma_\perp = \Gamma \cup \{\perp\}$, where $\perp$ is a special symbol, and assume without loss of generality that every transition in $\mathcal{P}$ that manipulates the counter does not affect the stack. Afterwards we will argue that the algorithm terminates within exponentially many iterations.

1. We start with $k = 0$. We construct a graph G_0 that consists of nodes $\langle p, \perp\rangle$ where p is a state of $\mathcal{P}$. Moreover, G_0 contains the corresponding transitions of $\mathcal{P}$ that do not manipulate the stack. In particular, for every transition $(p, i, \varepsilon, q) \in T$ we have an edge $\langle p, \perp\rangle \xrightarrow{i} \langle q, \perp\rangle$ in G_0. We use Lemma 5.7 to compute γ^{G_0} and δ^{G_0}, and report that, for all states p and q, we have $\gamma_0(p, q) = \gamma^{G_0}(\langle p, \perp\rangle, \langle q, \perp\rangle)$ and $\delta_0(p) = \delta^{G_0}(\langle p, \perp\rangle)$.
2. We repeat the following until γ^{G_k} and δ^{G_k} have converged. We construct a graph G_k as follows. For every state p and every $\sigma \in \Gamma_\perp$, we have a node $\langle p, \sigma\rangle$ in G_k. We then do the following.
   a. Let $\delta^{G_{k-1}}(\langle p, \perp\rangle) = c$. We insert a node $\langle p', \sigma\rangle$, and if $-\infty < c$, we insert two edges manipulating the counter $\langle p, \sigma\rangle \xrightarrow{c} \langle p', \sigma\rangle$ and $\langle p', \sigma\rangle \xrightarrow{-c+1} \langle p, \sigma\rangle$.
   b. For every state q, let $\gamma^{G_{k-1}}(\langle p, \perp\rangle, \langle q, \perp\rangle) = (a, b)$. If $-\infty < a$, we insert a node $\langle p, q, \sigma\rangle$ and two edges manipulating the counter $\langle p, \sigma\rangle \xrightarrow{a} \langle p, q, \sigma\rangle$ and $\langle p, q, \sigma\rangle \xrightarrow{-a+b'} \langle q, \sigma\rangle$, where $b' = b$ if $b < \omega$ and $b' = 0$ otherwise.
3. Finally, for each stack letter $\sigma \in \Gamma$, we connect nodes of the form $\langle p, \perp\rangle$ and $\langle q, \sigma\rangle$ using the transitions of $\mathcal{P}$ that manipulate the stack. That is, for every transition $(p, 0, \sigma, q) \in T$, we insert an edge $\langle p, \perp\rangle \to \langle q, \sigma\rangle$, and for every transition $(p, 0, \overline{\sigma}, q) \in T$, we insert an edge $\langle p, \sigma\rangle \to \langle q, \perp\rangle$.
4. We use Lemma 5.7 to compute γ^{G_k} and δ^{G_k}, and report that, for all states p and q, we have $\gamma_k(p, q) = \gamma^{G_k}(\langle p, \perp\rangle, \langle q, \perp\rangle)$ and $\delta_k(p) = \delta^{G_k}(\langle p, \perp\rangle)$.

We first argue that the graphs G_k consist of strongly connected components, and thus Lemma 5.7 is applicable.

▶ **Lemma 5.8.** *For all $k \in \mathbb{N}$, every connected component of G_k is strongly connected.*

Given some $\sigma \in \Gamma_\perp$ and $k \geq 1$, we denote by $G_k \downharpoonleft \sigma$ the subgraph of G_k induced by the nodes whose last element is σ. It follows directly from the construction of G_k that, for every pair of states p, q of $\mathcal{P}$ and $\sigma \in \Gamma_\perp$, for every path P that goes from $\langle p, \sigma\rangle$ to $\langle q, \sigma\rangle$ and is contained in $G_k \downharpoonleft \sigma$, there is a path $H \in \{p \rightsquigarrow q\}_{k-1}$ with $P \leq H$. In turn, this implies that the summary functions γ^{G_k} and δ^{G_k} are always dominated by paths in $\mathcal{P}$ of stack height at most k, i.e., for all states p and q, we have $\gamma^{G_k}(\langle p, \perp\rangle, \langle q, \perp\rangle) \leq \gamma_k(p, q)$ and $\delta^{G_k}(\langle p, \perp\rangle) \leq \delta_k(p)$ for all $k \in \mathbb{N}$. The following lemma states that γ^{G_k} and δ^{G_k} compute γ_k and δ_k exactly, by arguing that γ^{G_k} and δ^{G_k} also dominate all paths of $\mathcal{P}$ with stack height at most k. In turn, this establishes the correctness of the algorithm.

▶ **Lemma 5.9.** *For all $k \in \mathbb{N}$ and states $p, q \in Q$, we have $\gamma_k(p, q) = \gamma^{G_k}(\langle p, \perp\rangle, \langle q, \perp\rangle)$ and $\delta_k(p) = \delta^{G_k}(\langle q, \perp\rangle)$.*

Thus, to decide whether p covers $(q, 0)$, we saturate γ_k and δ_k and check whether $\gamma_k(p, q) = (0, \_)$. The termination and complexity of the algorithm follows from the boundedness of the finite values of γ_k and δ_k (Lemma 5.3, Lemma 5.4 and Lemma 5.6), which concludes Lemma 5.2.

▶ **Lemma 5.10.** *The above algorithm terminates and uses polynomial space.*

Finally, note that if we have a polynomial bound on the stack height, then the saturation procedure runs in polynomial time, which also leads to reachability in polynomial time (a closer analysis yields an $O(n^3)$ bound per iteration). In particular, the PSPACE-hardness proof for 1-dimensional *directed* PVASS from [15] cannot be directly transferred to bidirected PVASS: The 1-PVASS constructed in [15] has a polynomially bounded stack height. See the full version [20] for details on how exactly the reduction fails. Without a bound on the stack height, the saturation might indeed take exponential time: There are 1-dimensional bidirected PVASS on which shortest coverability witnesses require an exponential stack height (see the full version [20] for an example).

6 Tower lower bound

In this section, we show that reachability in bidirected PVASS is TOWER-hard with respect to elementary reductions, and k-EXPSPACE-hard in dimension $2k + 6$. Recall that TOWER is the class of all problems computable by deterministic Turing machines in time (or space) bounded by a tower of exponentials of elementary height.

Lower bound for directed PVASS. We first recall the TOWER-hardness proof by Lazić and Totzke [33] for reachability in directed PVASS. They reduce the $\exp^n(1)$-bounded halting problem for counter programs of size n, which is TOWER-complete with respect to elementary reductions (which allow us to replace the parameter n with an arbitrary elementary function $e(n)$) [19]. A *counter program* is a finite sequence of commands which manipulate non-negative counters, initially set to zero. The commands include increments $x := x + 1$, decrements $x := x - 1$, conditionals **if** $x = 0$ **then goto** L_1 **else goto** L_2 (where L_1 and L_2 are line numbers), and **halt**. If a counter of value 0 is decremented, the program aborts. The $\exp^n(1)$-bounded halting problem asks whether given a counter program of size n, starting from the first command and all counters set to zero, a command **halt** can be reached using a run during which all counters are bounded by $\exp^n(1)$ and are all zero at the end.

As in most lower bounds for vector addition systems and their extensions, for each counter x we store a complement counter $\bar{x}$, maintaining the invariant $x + \bar{x} = B$ for some (large) bound B. This can be achieved by complementing every in/decrement of x by a de/increment of $\bar{x}$, and vice versa. Then, a zero test **if** $x = 0$ **then goto** L_1 **else goto** L_2 can be replaced by guessing whether $x = 0$ and $x \neq 0$: In the former case we add and subtract B to x and continue with L_1. In the latter case we add and subtract B to $\bar{x}$ and continue with L_2.

The challenge is to implement the addition (and subtraction) with a large number B, here $B = \exp^n(1)$, using a polynomially large system. Suppose we have counters $c_1, \dots, c_n$ with complement counters $\bar{c}_1, \dots, \bar{c}_n$ satisfying $c_k + \bar{c}_k = \exp^k(1)$ for all k. Lazić and Totzke [33] show how to construct for all $k = 1, \dots, n$ a poly(k)-sized PVASS that adds $\exp^k(1)$ to c_k. It operates by pushing exactly $\exp^{k-1}(1)$ many zeros to the stack, repeatedly incrementing the $\exp^{k-1}(1)$-bit binary counter on the stack, while simultaneously decrementing c_k, and finally popping exactly $\exp^{k-1}(1)$ many ones from the stack. Observe that the operations on the stack can be implemented with the help of c_{k-1} that can be in/decremented by $\exp^{k-1}(1)$ by induction hypothesis. Before simulating the counter program, each complement counter $\bar{c}_k$ has to be set to $\exp^k(1)$, which can be done in a similar fashion.

Simulation by bidirected systems. In the following we will make the above construction outlined by Lazić and Totzke [33] explicit and show that the simulation is correct even after making the PVASS bidirected. To this end we need the following argument already found in Post's undecidability proof of the word problem for Thue systems [46, Lemma II]. Consider a deterministic transition system where the final state does not have any outgoing transitions. To such a system we now add reverse edges to make it bidirected. Clearly, any original run is present in the bidirected system. Conversely, consider a run to the final state in the bidirected system, which may use the new reverse edges. It cannot end on a reverse edge, since there is no outgoing forward edge from the final state. So as long as the run contains reverse edges, at least one of these edges must be followed by one in the forward direction. Let us call them $p \xrightarrow{\bar{a}} q$ and $q \xrightarrow{b} r$. As the original system was deterministic q has exactly one outgoing edge, and hence $(q \xrightarrow{b} r) = (q \xrightarrow{a} p)$. Since the effects of $\bar{a}$ and b cancel out, we can omit both of them from the run. Iterating this argument eventually yields a run with no reverse edges. It follows that adding reverse edges to a fully deterministic system does not change its reachability set (this was originally shown by Mayr and Meyer [43] for their proof of EXPSPACE-hardness of reachability for bidirected VAS).

The construction of Lazić and Totzke is not fully deterministic. However, it only uses very restricted nondeterminism that will not impact our simulation of the counter machine.

Gadget construction. Since we also want to show a k-EXPSPACE lower bound for dimension $2k+6$, we use a slightly more refined analysis: We will assume that two numbers k and n are given as input and construct a system that simulates counters bounded by $\exp^k(n)$ instead of $\exp^k(1)$ as in Lazić and Totzke.

In the following a *gadget* G consists of a PVASS and two distinguished terminal states s and t. We consider vectors $\boldsymbol{x} \in \mathbb{N}^{2k}$ where the first k components are viewed the values of k counters $c_1, \ldots, c_k$ and the last k components are the values of k complementary counters $\bar{c}_1, \ldots, \bar{c}_k$. Without further mention, any update on a counter c is always understood with complementary update on $\bar{c}$ so that the sums $c_i + \bar{c}_i$ remain constant.

Given two numbers k, n (in unary), we will inductively construct a gadget G_k with stack alphabet Γ_k. This gadget will allow us to add $\exp^k(n)$ to a counter. The gadget's size will grow exponentially in k (and polynomially in n), and later, we improve the construction to grow only polynomially in k. The gadget G_1 simply increments c_1 by 2^n. Assuming G_{k-1} is already constructed, we construct the gadget G_k. The gadget $\bar{G}_{k-1}$ is obtained from G_{k-1} by reversing all transitions, and interchanging its terminal states. Its behavior is inverse to that of G_{k-1}, as it subtracts $\exp^{k-1}(n)$ from c_{k-1}. Let $Z_{k-1} = G_{k-1} \circ \bar{G}_{k-1}$ be the gadget obtained by composing G_{k-1} with $\bar{G}_{k-1}$, which is a zero test of c_{k-1}. We can naturally view G_{k-1}, $\bar{G}_{k-1}$ and Z_{k-1} as gadgets with $2k$ counters, where c_k and $\bar{c}_k$ are untouched. The gadget G_k is displayed in Figure 1 where `0` and `1` are fresh stack symbols and Inc_k is a subprocedure which increments the binary counter on the stack.

To prove correctness of the gadget G_k we need a bit of notation. For brevity we write $[x_1, \ldots, x_k]$ for $(x_1, \ldots, x_k, \exp^1(n) - x_1, \ldots, \exp^k(n) - x_k)$. Our gadgets will always assume that the "lower" counters c_j are set to zero and that the invariant is satisfied. A counter vector of the form $[0, \ldots, 0, x_i, \ldots, x_k]$ is called *i-initialized*. Moreover, we call a run $(s, \boldsymbol{u}, w) \xrightarrow{*} (t, \boldsymbol{v}, w')$ in a gadget *i-initialized* if either $\boldsymbol{u}$ or $\boldsymbol{v}$ is i-initialized.

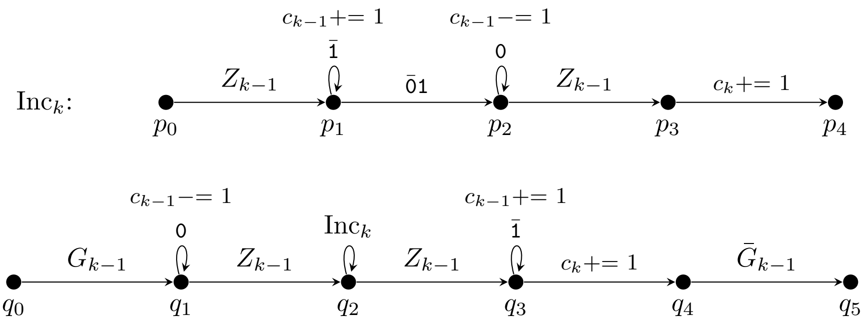

Figure 1 Gadgets Inc$_k$ and G_k.

▶ **Proposition 6.1.** *The k-initialized runs in G_k from q_0 to q_5 are precisely the runs*

$$(q_0, [0, \ldots, 0], w) \xrightarrow{*}_{G_k} (q_5, [0, \ldots, 0, \exp^k(n)], w) \quad \textit{for } w \in \Gamma_{k-1}^*.$$

Next we will analyse the bidirected version of G_k. In order to distinguish the original transitions from the reverse transitions we define for a PVASS G the relations $\leftrightarrow_G = \rightarrow_G \cup \leftarrow_G$ and $\overset{*}{\leftrightarrow}_G$, denoting the reflexive transitive closure of $\leftrightarrow_G$. Similarly to the argument by Post [46, Lemma II], we can prove the following:

▶ **Proposition 6.2.** *Let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^{2k}$ where $\boldsymbol{u}$ or $\boldsymbol{v}$ is k-initialized.*

- *If $(q_0, \boldsymbol{u}, w) \overset{*}{\leftrightarrow}_{G_k} (q_5, \boldsymbol{v}, w')$ then $(q_0, \boldsymbol{u}, w) \xrightarrow{*}_{G_k} (q_5, \boldsymbol{v}, w')$.*
- *If $q \in \{q_0, q_5\}$ and $(q, \boldsymbol{u}, w) \overset{*}{\leftrightarrow}_{G_k} (q, \boldsymbol{v}, w')$ then $\boldsymbol{u} = \boldsymbol{v}$ and $w = w'$.*

We need to reduce the size of G_k so that it can be constructed in time polynomial in k. Since G_k uses ten copies of the subgadget G_{k-1} (each zero test Z_{k-1} uses two copies of G_{k-1}), we cannot simply insert G_{k-1} by copying it, as this would induce exponential growth of the number of states of our system. Instead, we instantiate each gadget G_{k-1} once. Then, whenever a gadget would be used between two states p, q, we push a fresh stack symbol $t_{p,q}$ and move to G_{k-1}. When exiting G_{k-1} we pop $t_{p,q}$ and return to q. Since this symbol is unique for every pair of states, it uniquely determines where we can leave the gadget to, even if there are multiple incoming and outgoing transitions at the gadget G_{k-1}. Finally, one can verify that Proposition 6.2 still holds for this adapted version of G_k.

Simulating the counter program. We are ready to finish the lower bound proof. We are given a counter program of size n with three counters x_1, x_2, x_3 and want to reduce the $\exp^{k+1}(n)$-bounded halting problem to the reachability problem for bidirected PVASS using $2k+6$ counters. To this end, we construct the gadget G_{k+1} three times: Each of these three instances has, instead of c_{k+1} (and its complement), a counter x_i (and its complement) for some $i \in \{1, 2, 3\}$. However, the three instances of G_{k+1} share the counters $c_1, \ldots, c_k$ (and their complements). Thus, in total, we have $2 \cdot k + 2 \cdot 3 = 2k + 6$ counters. If k is fixed, this yields our k-EXPSPACE lower bound ([19]). If k is part of the input, the problem becomes TOWER-complete. We start by initializing the complement counters $\bar{c}_1, \ldots, \bar{c}_k$ in sequence, using variants of the gadgets G_i that (i) operate on the balance counter $\bar{c}_i$ instead of c_i, (ii) do not decrement c_i when incrementing $\bar{c}_i$, and (iii) operate on the lower $i-1$ counters as normal. Similarly we initialize $\bar{x}_1, \bar{x}_2, \bar{x}_3$ to $\exp^{k+1}(n)$. Finally, in order to have an all-zero configuration in the final state, we de-initialize these counters before entering the final state.

Increments and decrements in the counter program are directly translated into counter updates in the PVASS. A conditional **if** $x_i = 0$ **then goto** L_1 **else goto** L_2 is replaced by a nondeterministic guess of whether $x_i = 0$ or $x_i \neq 0$, verifying this (in)equality, and jumping to L_1 or L_2. Here we use variants of the zero tests $Z_{k+1} = G_{k+1} \circ \bar{G}_{k+1}$ which on their highest level operate on x and $\bar{x}$ (instead of c_{k+1} and $\bar{c}_{k+1}$). The question of reachability of **halt** is then a reachability instance on the bidirected version of the PVASS.

If the counter program halts then we can find a corresponding computation in the PVASS. Conversely, consider a successful run of the bidirected PVASS which uses a minimal number of reverse transitions. By Proposition 6.2 we can assume that no gadget G_{k+1} and $\bar{G}_{k+1}$ (and their variants) is entered and exited through the same terminal state. Furthermore, any subrun passing through such a gadget can be assumed to use only forward transitions. Hence the only reverse transitions remaining are from increments or decrements. Observe that the last occurrence of such a reverse transition $\bar{\tau}$ must be followed by its corresponding forward transition τ. Hence we can cancel τ with $\bar{\tau}$, contradiction.

7 Conclusion

We have shown that the reachability problem in bidirected pushdown VASS is decidable, with an Ackermann upper bound and a TOWER lower bound. Moreover, in the one-dimensional case, the problem is in PSPACE, whereas P-hardness was shown in [21]. Thus, the exact complexity, both in the general and the one-dimensional case, remains open.

Another direction for future research is to study bidirected versions of other infinite-state models. For example, pushdown VASS are the simplest level in a hierarchy of infinite-state models for which decidability of the reachability problem is open [53]. Perhaps the techniques from this paper can be applied to show decidability of all levels in the bidirected setting.

References

1. Mohamed Faouzi Atig and Pierre Ganty. Approximating Petri net reachability along context-free traces. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 152–163. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.152.
2. Thomas Becker, Volker Weispfenning, and Heinz Kredel. *Gröbner bases - a computational approach to commutative algebra*, volume 141 of *Graduate texts in mathematics*. Springer, 1993.
3. Rémi Bonnet. *Theory of well-structured transition systems and extended vector-addition systems*. PhD thesis, ENS Cachan, France, 2013. Thèse de doctorat.
4. Rémi Bonnet, Alain Finkel, Jérôme Leroux, and Marc Zeitoun. Model checking vector addition systems with one zero-test. *Log. Methods Comput. Sci.*, 8(2), 2012. doi:10.2168/LMCS-8(2:11)2012.
5. Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997. doi:10.1007/3-540-63141-0_10.
6. Zakaria Bouziane and Alain Finkel. Cyclic Petri net reachability sets are semi-linear effectively constructible. In *Second International Workshop on Verification of Infinite State Systems, Infinity 1997, Bologna, Italy, July 11-12, 1997*, volume 9 of *Electronic Notes in Theoretical Computer Science*, pages 15–24. Elsevier, 1997. doi:10.1016/S1571-0661(05)80423-2.
7. Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal*. PhD thesis, Universität Innsbruck, 1965.
8. Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. Optimal Dyck Reachability for Data-Dependence and Alias Analysis. *Proc. ACM Program. Lang.*, 2(POPL), December 2018. doi:10.1145/3158118.

9 Swarat Chaudhuri. Subcubic algorithms for recursive state machines. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 159–169. ACM, 2008. doi:10.1145/1328438.1328460.

10 David C Cooper. Theorem proving in arithmetic without multiplication. *Machine intelligence*, 7(91-99):300, 1972.

11 Wojciech Czerwiński, Slawomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 24–33. ACM, 2019. doi:10.1145/3313276.3316369.

12 Wojciech Czerwiński and Łukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1229–1240. IEEE, 2021. doi:10.1109/FOCS52979.2021.00120.

13 Thomas W Dubé. The structure of polynomial ideals and Gröbner bases. *SIAM Journal on Computing*, 19(4):750–773, 1990. doi:10.1137/0219053.

14 Samuel Eilenberg and M. P. Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13(2):173–191, 1969. doi:10.1016/0021-8693(69)90070-2.

15 Matthias Englert, Piotr Hofman, Slawomir Lasota, Ranko Lazić, Jérôme Leroux, and Juliusz Straszynski. A lower bound for the coverability problem in acyclic pushdown VAS. *Inf. Process. Lett.*, 167:106079, 2021. doi:10.1016/j.ipl.2020.106079.

16 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and Primitive-Recursive Bounds with Dickson's Lemma. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 269–278. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.39.

17 Alain Finkel, Jérôme Leroux, and Grégoire Sutre. Reachability for two-counter machines with one test and one reset. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.31.

18 Alain Finkel and Grégoire Sutre. Decidability of reachability problems for classes of two counters automata. In *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 346–357. Springer, 2000. doi:10.1007/3-540-46541-3_29.

19 Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter machines and counter languages. *Mathematical systems theory*, 1968. doi:10.1007/BF01694011.

20 Moses Ganardi, Rupak Majumdar, Andreas Pavlogiannis, Lia Schütze, and Georg Zetzsche. Reachability in bidirected pushdown VASS. Full version of this paper. arXiv:2204.11799.

21 Moses Ganardi, Rupak Majumdar, and Georg Zetzsche. The complexity of bidirected reachability in valence systems. To appear in Proc. of the Thirty-Seventh Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2022). arXiv:2110.03654.

22 Jan Grabowski. An algorithm to identify slices, with applications to vector replacement systems. In *International Conference on Fundamentals of Computation Theory (FCT 1981)*, pages 425–432. Springer, 1981. doi:10.1007/3-540-10854-8_46.

23 Christoph Haase. A survival guide to Presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. doi:10.1145/3242953.3242964.

24 Matthew Hague and Anthony Widjaja Lin. Model checking recursive programs with numeric data types. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 743–759. Springer, 2011. doi:10.1007/978-3-642-22110-1_60.

25 Tero Harju, Oscar H. Ibarra, Juhani Karhumäki, and Arto Salomaa. Some decision problems concerning semilinearity and commutation. *J. Comput. Syst. Sci.*, 65(2):278–294, 2002. doi:10.1006/jcss.2002.1836.

26 Nevin Heintze and David A. McAllester. On the cubic bottleneck in subtyping and flow analysis. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science (LICS), Warsaw, Poland, June 29 - July 2, 1997*, pages 342–351. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614960.

27 Yoram Hirshfeld. *Congruences in commutative semigroups*. LFCS, Department of Computer Science, University of Edinburgh Edinburgh, 1994.

28 John E Hopcroft and Jeffrey D Ullman. *Introduction to Automata Theory, Languages and Computation*. Adison-Wesley, Reading, Mass, 1979.

29 Dung T. Huynh. A superexponential lower bound for Gröbner bases and Church-Rosser commutative Thue systems. *Inf. Control.*, 68(1-3):196–206, 1986. doi:10.1016/S0019-9958(86)80035-3.

30 Jarkko Kari. Reversible cellular automata: From fundamental classical results to recent developments. *New Gener. Comput.*, 36(3):145–172, 2018. doi:10.1007/s00354-018-0034-6.

31 Adam Husted Kjelstrøm and Andreas Pavlogiannis. The decidability and complexity of interleaved bidirected Dyck reachability. *Proc. ACM Program. Lang.*, 6(POPL):1–26, 2022. doi:10.1145/3498673.

32 Ulla Koppenhagen and Ernst W. Mayr. The complexity of the coverability, the containment, and the equivalence problems for commutative semigroups. In *Fundamentals of Computation Theory, 11th International Symposium, FCT '97, Kraków, Poland, September 1-3, 1997, Proceedings*, volume 1279 of *Lecture Notes in Computer Science*, pages 257–268. Springer, 1997. doi:10.1007/BFb0036189.

33 Ranko Lazić and Patrick Totzke. *What Makes Petri Nets Harder to Verify: Stack or Data?*, pages 144–161. Springer International Publishing, 2017. doi:10.1007/978-3-319-51046-0_8.

34 Jérôme Leroux. Vector addition system reachability problem: a short self-contained proof. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 307–316. ACM, 2011. doi:10.1145/1926385.1926421.

35 Jérôme Leroux. Vector addition system reversible reachability problem. *Log. Methods Comput. Sci.*, 9(1), 2013. doi:10.2168/LMCS-9(1:5)2013.

36 Jérôme Leroux. Distance between mutually reachable Petri net configurations. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.47.

37 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1241–1252. IEEE, 2021. doi:10.1109/FOCS52979.2021.00121.

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

39 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 324–336. Springer, 2015. doi:10.1007/978-3-662-47666-6_26.

40 Yuanbo Li, Qirun Zhang, and Thomas W. Reps. Fast graph simplification for interleaved Dyck-reachability. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 780–793. ACM, 2020. doi:10.1145/3385412.3386021.

41 Yuanbo Li, Qirun Zhang, and Thomas W. Reps. On the complexity of bidirected interleaved Dyck-reachability. *Proc. ACM Program. Lang.*, 5(POPL):1–28, 2021. `doi:10.1145/3434340`.

42 Markus Lohrey and Benjamin Steinberg. An automata theoretic approach to the generalized word problem in graphs of groups. *Proceedings of the American Mathematical Society*, 138(2):445–453, 2010. `doi:10.1090/S0002-9939-09-10126-0`.

43 Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in mathematics*, 46(3):305–329, 1982. `doi:10.1016/0001-8708(82)90048-2`.

44 Derek C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *J. Comput. Syst. Sci.*, 16(3):323–332, 1978. `doi:10.1016/0022-0000(78)90021-1`.

45 Laurent Pierre. Rational indexes of generators of the cone of context-free languages. *Theoretical Computer Science*, 95(2):279–305, 1992. `doi:10.1016/0304-3975(92)90269-L`.

46 Emil L. Post. Recursive unsolvability of a problem of Thue. *J. Symb. Log.*, 12(1):1–11, 1947. `doi:10.2307/2267170`.

47 Klaus Reinhardt. Reachability in Petri nets with inhibitor arcs. *Electron. Notes Theor. Comput. Sci.*, 223:239–264, 2008. `doi:10.1016/j.entcs.2008.12.042`.

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

49 Sylvain Schmitz. *Algorithmic Complexity of Well-Quasi-Orders. (Complexité algorithmique des beaux pré-ordres)*. Habilitation thesis, École normale supérieure Paris-Saclay, 2017. URL: `https://tel.archives-ouvertes.fr/tel-01663266`.

50 Sylvain Schmitz and Georg Zetzsche. Coverability is undecidable in one-dimensional pushdown vector addition systems with resets. In *Reachability Problems - 13th International Conference, RP 2019, Brussels, Belgium, September 11-13, 2019, Proceedings*, volume 11674 of *Lecture Notes in Computer Science*, pages 193–201. Springer, 2019. `doi:10.1007/978-3-030-30806-3_15`.

51 Guoqing Xu, Atanas Rountev, and Manu Sridharan. Scaling CFL-reachability-based points-to analysis using context-sensitive must-not-alias analysis. In *ECOOP 2009 - Object-Oriented Programming, 23rd European Conference, Genoa, Italy, July 6-10, 2009. Proceedings*, volume 5653 of *Lecture Notes in Computer Science*, pages 98–122. Springer, 2009. `doi:10.1007/978-3-642-03013-0_6`.

52 Dacong Yan, Guoqing Xu, and Atanas Rountev. Demand-driven context-sensitive alias analysis for Java. In *Proceedings of the 20th International Symposium on Software Testing and Analysis, ISSTA 2011, Toronto, ON, Canada, July 17-21, 2011*, pages 155–165. ACM, 2011. `doi:10.1145/2001420.2001440`.

53 Georg Zetzsche. The emptiness problem for valence automata over graph monoids. *Information and Computation*, 277, 2021. `doi:10.1016/j.ic.2020.104583`.

54 Qirun Zhang, Michael R. Lyu, Hao Yuan, and Zhendong Su. Fast algorithms for Dyck-CFL-reachability with applications to alias analysis. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, pages 435–446. ACM, 2013. `doi:10.1145/2491956.2462159`.

55 Qirun Zhang and Zhendong Su. Context-sensitive data-dependence analysis via linear conjunctive language reachability. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 344–358. ACM, 2017. `doi:10.1145/3009837.3009848`.