Scope-Bounded Reachability in Valence Systems

Aneesh K. Shetty
Department of Computer Science & Engineering IIT Bombay, India

S. Krishna
Department of Computer Science & Engineering IIT Bombay, India

Georg Zetzsche
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Abstract

Multi-pushdown systems are a standard model for concurrent recursive programs, but they have an undecidable reachability problem. Therefore, there have been several proposals to underapproximate their sets of runs so that reachability in this underapproximation becomes decidable. One such underapproximation that covers a relatively high portion of runs is *scope boundedness*. In such a run, after each push to stack i, the corresponding pop operation must come within a bounded number of visits to stack i.

In this work, we generalize this approach to a large class of infinite-state systems. For this, we consider the model of valence systems, which consist of a finite-state control and an infinite-state storage mechanism that is specified by a finite undirected graph. This framework captures pushdowns, vector addition systems, integer vector addition systems, and combinations thereof. For this framework, we propose a notion of scope boundedness that coincides with the classical notion when the storage mechanism happens to be a multi-pushdown.

We show that with this notion, reachability can be decided in PSPACE for every storage mechanism in the framework. Moreover, we describe the full complexity landscape of this problem across all storage mechanisms, both in the case of (i) the scope bound being given as input and (ii) for fixed scope bounds. Finally, we provide an almost complete description of the complexity landscape if even a description of the storage mechanism is part of the input.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Formal languages and automata theory

Keywords and phrases multi-pushdown systems, underapproximations, valence systems, reachability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.29

Related Version *Full version with proofs*: `https://arxiv.org/abs/2108.00963`

1 Introduction

Multi-pushdown systems are a natural model for recursive programs with threads that communicate via shared memory. Unfortunately, even safety verification (state reachability) is undecidable for this model [21]. However, by considering *underapproximations* of the set of all executions, it is still possible to discover safety violations. The first such underapproximation in the literature was *bounded context switching* [20]. Here, one only considers executions that switch between threads a bounded number of times. In terms of multi-pushdown systems, this places a bound on the number of times we can switch between stacks.

One underapproximation that covers a relatively large portion of all executions and still permits decidable reachability is *scope-boundedness* as proposed by La Torre, Napoli, and Parlato [23, 25]. Here, instead of bounding the number of context switches across the entire run, we bound the number of context switches *per letter on a stack* (i.e. procedure execution). More precisely, whenever we push a letter on some stack i, then we can switch back to stack i at most k times before we have to pop that letter again. This higher coverage of runs comes at the cost of higher complexity: While reachability with bounded context switching is NP-complete [12, 20], the scope-bounded reachability problem is PSPACE-complete (if the number of pushdowns or the scope bound is part of the input) [25].

Aside from multi-pushdown systems, there is a wide variety of other infinite-state models that are used to model program behaviours. For these, reachability problems are also sometimes undecidable or have prohibitively high complexity. For example, vector addition systems with states (VASS) is one of the most prominent models for concurrent systems, but its reachability problem has non-elementary complexity [8]. This raises the question of whether underapproximations for multi-pushdown systems can be interpreted in other infinite-state systems and what complexity would ensue.

The notion of bounded context switching has recently been generalized to a large class of infinite-state systems [19], in the framework of *valence systems over graph monoids*. These consist of a finite-state control that has access to a storage mechanism. The shape of this storage mechanism is described by a finite, undirected graph. By choosing an appropriate graph, one can realize many infinite-state models. Examples include (multi-)pushdown systems, VASS, integer VASS, but also combinations thereof, such as pushdown VASS [17] and sequential recursive Petri nets [16]. Under this notion, bounded context reachability is in NP for each graph, and thus each storage mechanism in the framework [19]. Moreover, the paper [19] presents some subclasses of graphs for which bounded context reachability has lower complexity (NL or P). However, the exact complexity of reachability under bounded context switching remains open in many cases, such as the path with four nodes [19].

Contribution. We present an *abstract notion of scope-bounded runs* for valence systems over graph monoids. As we show, this notion always leads to a reachability problem decidable in PSPACE. In particular, our notion applies to all infinite-state models mentioned above. Moreover, applied to multi-pushdown systems, it coincides with the notion of La Torre, Napoli, and Parlato.

We also obtain an almost complete complexity landscape of scope-bounded reachability. First, we show that if both (i) the graph Γ describing the storage mechanism and (ii) the scope bound k are part of the input, the problem is PSPACE-complete. Second, we study how the complexity depends on the employed storage mechanism. We show that for each Γ, the problem is either NL-complete, P-complete, or PSPACE-complete, depending on Γ (Corollary 4.2). Since the complexity drops below PSPACE only in extremely restricted cases, we also study the setting where the scope bound k is fixed. In this case, we show that the problem is either NL-complete or P-complete, depending on Γ (Corollary 4.4).

Finally, applying scope-boundedness to classes of infinite-state systems requires understanding the complexity if Γ is drawn from an infinite class of graphs. For example, for each fixed dimension d, there is a graph Γ_d such that valence automata over Γ_d correspond to VASS of dimension d. The class of *all* VASS (of arbitrary dimension), however, corresponds to valence automata over all cliques. Thus, we also study scope-bounded reachability if Γ is restricted to a class of graphs $\mathcal{G}$. Under a mild assumption on $\mathcal{G}$, we again obtain a complexity trichotomy of NL-, P-, or PSPACE-completeness, both for k as input (Theorem 4.1) and for fixed k (Theorem 4.3). In fact, all results mentioned above follow from these general results.

Related work. Similar in spirit to our work are the lines of research on systems with bounded tree-width by Madhusudan and Parlato [18] and on bounded split-width by Aiswarya [6]. In these settings, the storage mechanism is represented as a class of possible matching relations on the positions of a computation. Then, under the assumption that the resulting *behavior graphs* have bounded tree-width or split-width, respectively, there are general decidability results. In particular, decidability of scope-bounded reachability in multi-pushdown systems has been deduced via tree-width [26] and via split-width [7]. Different from underapproximations based on bounded tree-width or split-width, our framework includes multi-counter systems (such as VASS or integer VASS), but also counters nested in stacks. While VASS can be seen as special cases of multi-pushdown systems, our framework allows us, e.g. to study the complexity of scope-bounded reachability if the storage mechanism is restricted to multi-counters. On the other hand, while tree-width and split-width can be considered for queues [18, 1], they cannot be realized as storage mechanisms in valence systems.

Furthermore, after their introduction [23] scope-bounded multi-pushdown systems have been studied in terms of accepted languages [24], temporal logic model checking [26, 3]. Moreover, scope-boundedness has been studied in the timed setting [2],[4].

Over the last decade, the framework of valence automata over graph monoids has been used to study how several types of analysis are impacted by the choice of storage mechanism. For example: For which storage mechanisms (i) can silent transitions be algorithmically eliminated? [27]; (ii) do we have a Parikh's theorem [5], (iii) is (general) reachability decidable [31]; (iv) is first-order logic with reachability decidable? [10]; (v) can downward closures be computed effectively? [28].

Details of all proofs can be found in the full version of the paper.

2 Preliminaries

In this section, we recall the basics of valence systems over graph monoids [29].

Graph Monoids. This class of monoids accommodate a variety of storage mechanisms. They are defined by undirected graphs without parallel edges $\Gamma = (V, I)$ where V is a finite set of vertices and $I \subseteq \{e \subseteq V \mid 1 \leq |e| \leq 2\}$ is a finite set of undirected edges, which can be self-loops. Thus, if $\{v\} \in I$, we say that v is *looped*; otherwise, v is *unlooped*. The edge relation is also called an *independence relation*. We also write uIv for $\{u, v\} \in I$. A subset $U \subseteq V$ is a *clique* if uIv for any two distinct $u, v \in U$. If in addition, all $v \in U$ are looped, then U is a *looped clique*. If U is a clique and all $v \in U$ are unlooped, then U is an *unlooped clique*. We say that $U \subseteq V$ is an *anti-clique* if we do not have uIv for any distinct $u, v \in U$. Given the graph, we define a monoid as follows. We have the alphabet $X_\Gamma = \{v^+, v^- \mid v \in V\}$, where we write xIy for $x, y \in X_\Gamma$ if for some $u, v \in V$, we have $x \in \{u^+, u^-\}$, $y \in \{v^+, v^-\}$, $x \neq y$, and uIv. Moreover, $\equiv_\Gamma$ is the smallest congruence on X_Γ^* with $v^+v^- \equiv_\Gamma \varepsilon$ for $v \in V$ and $xy \equiv_\Gamma yx$ for xIy. Here, ε denotes the empty word. Thus, if v has a self-loop, then $v^-v^+ \equiv_\Gamma \varepsilon$. We define the monoid $\mathbb{M}\Gamma := X_\Gamma^* / \equiv_\Gamma$.

Valence Systems. Graph monoids are used in valence systems, which are finite automata whose edges are labeled with elements of a monoid. Then, a run is considered valid if the product of the monoid elements is the neutral element. Here, we only consider the case where the monoid is of the form $\mathbb{M}\Gamma$, so we define the concept directly for graphs.

Given a graph Γ, a valence system $\mathcal{A}$ over Γ consists of a finite set of states Q, and a finite transition relation $\rightarrow \subseteq Q \times X_\Gamma^* \times Q$. A *configuration* of $\mathcal{A}$ is a tuple (q, w) where $q \in Q$, $w \in X_\Gamma^*$ is the sequence of storage operations executed so far. From a configuration (q_1, u), on a transition $q_1 \xrightarrow{v} q_2$, we reach the configuration (q_2, uv). A run of $\mathcal{A}$ is a sequence of transitions. The *reachability problem* in valence systems is the following: Given states q_{init} and q_{fin}, is there a run from (q_{init}, ε) that reaches (q_{fin}, w) for some $w \in X_\Gamma^*$ with $w \equiv_\Gamma \varepsilon$?

Many classical storage types can be realized with graph monoids. Consider $\bar{\Gamma}_3 = (V, I)$ in Figure 1. We have $I = \{\{a, c\}, \{b, c\}, \{c\}\}$. For $w \in X_{\bar{\Gamma}_3}^*$ we have $w \equiv_{\bar{\Gamma}_3} \varepsilon$ if and only if two conditions are met: First, if we project to $\{a^+, a^-, b^+, b^-\}$, then the word corresponds to a sequence of push- and pop-operations that transform the empty stack into the empty stack. Here, x^+ corresponds to pushing x, and x^- to popping x, for $x \in \{a, b\}$. Second, the number of c^+ is the same as the number of c^- in w. Thus, valence automata over $\bar{\Gamma}_3$ can be seen as pushdown automata that have access to a $\mathbb{Z}$-valued counter. Similarly, the storage mechanism of Γ_2 in Figure 1 is a stack, where each stack entry is not a letter, but contains two $\mathbb{N}$-valued counters. A push (c^+) starts a new stack entry and a pop (c^-) is only possible if the topmost two counters are zero. For more examples and explanation, see [30].

▶ **Example 2.1** (Example storage mechanisms). Let us mention a few particular (classes of) graphs and how they correspond to infinite-state systems. In the following, the *direct product* of two graphs Γ and Δ is the graph obtained by taking the disjoint union of Γ and Δ and adding an edge between each vertex from Γ and each vertex from Δ.

Pushdown For $s \in \mathbb{N}$, let P_s be the graph on s vertices without edges. Then valence automata over P_s correspond to pushdown systems with s stack symbols.

Multi-pushdown Let $\mathsf{MP}_{r,s}$ be the direct product of r disjoint copies of P_s. Then valence systems over $\mathsf{MP}_{r,s}$ correspond to multi-pushdown systems with r stacks, each of which has s stack symbols. In Figure 1, the induced subgraph of graph Γ_1 on $\{b_1, b_2, b_3, c_1, c_2, c_3\}$ represents $\mathsf{MP}_{2,3}$.

VASS If UC_d is an unlooped clique with d vertices, then valence systems over UC_d correspond to d-dimensional vector addition systems with states.

Integer VASS If LC_d is a looped clique with d vertices, then valence systems over LC_d correspond to d-dimensional integer VASS.

Pushdown VASS If UC_d^- is the graph obtained from UC_{d+2} by removing a single edge, then valence systems over UC_d^- correspond to d-dimensional pushdown VASS.

3 Scope-bounded runs in valence systems

In this section, we introduce our notion of bounded scope to valence systems over arbitrary graph monoids. For each of the used concepts, we will explain how they relate to the existing notion of scope-boundedness for multi-pushdown systems. Fixing $\Gamma = (V, I)$ as before, first we introduce some preliminary notations and definitions.

Dependent sets and contexts. Recall that valence systems over the graph $\mathsf{MP}_{r,s}$ realize a storage consisting of r pushdowns, each with s stack symbols. The graph $\mathsf{MP}_{r,s}$ is a direct product of r-many disjoint anti-cliques, each with s vertices. Here, each anti-clique corresponds to a pushdown with s stack symbols: For a vertex v in such an anti-clique, the symbol v^+ is the push operation for this stack symbol, and v^- is its pop operation.

In a multi-pushdown system, a run is naturally decomposed into contexts, where each context is a sequence of operations belonging to one stack. In [19], the notion of context was generalized to valence systems as follows. A set $U \subseteq V$ is called *dependent* if it does not contain distinct vertices $u_1, u_2 \in V$ such that $u_1 I u_2$. A set of operations $Y \subseteq X_\Gamma$ is *dependent* if its underlying set of vertices $\{v \in V \mid v^+ \in Y \text{ or } v^- \in Y\}$ is dependent. A computation is called *dependent* if the set of operations occurring in it is dependent. A dependent computation is also called a *context*. In Γ_1 of Figure 1, contexts can be formed over $\{b_1, b_2, b_3\}$, $\{c_1, c_2, c_3\}$ and $\{a\}$.

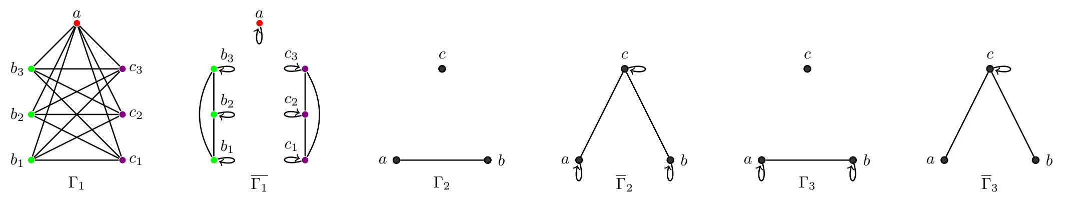

Figure 1 The storage mechanism of Γ_1 is 2 stacks and one partially blind counter. Symbols of the same stack are weakly dependent. In the storage mechanism of Γ_2, a, b, c are weakly dependent.

Context decomposition. Note that a word $w \in X_\Gamma^*$ need not have a unique decomposition into contexts. For example, for Γ_2 in Figure 1, the word $a^+c^+b^+$ can be decomposed as $(a^+c^+)b^+$ and as $a^+(c^+b^+)$. Therefore, we now define a canonical decomposition into contexts, which decomposes the word from left to right. Formally, the *canonical context decomposition* of a computation $w \in X_\Gamma^+$ (that is, $|w| > 0$) is defined inductively. If w is over a dependent set of operations, then w is a single context. Otherwise, find the maximal, non-empty prefix w_1 of w over a dependent set of operations. The canonical decomposition of w into contexts is then $w = w_1 w_2 \dots w_m$ where $w_2 \dots w_m$ is the decomposition of the remaining word into contexts. In the following, unless explicitly specified otherwise, when we mention the contexts of a word, we always mean those in the canonical decomposition. Observe that in the case of $\mathsf{MP}_{r,s}$, this is exactly the decomposition into contexts of multi-pushdown systems.

Reductions. Given a computation $w = a_1 \cdots a_n$ where each $a_i \in X_\Gamma$, we identify each operation a_i with its position. We denote by $w[i]$ the ith operation of w, hence $w[i] = a_i$. A *reduction* of w is a finite sequence of applications of the following rewriting rules.

(R1) $w'.w[x].w[y].w'' \mapsto_{red} w'w''$, applicable if $w[x] = o^+, w[y] = o^-$ for some o.

(R2) $w'.w[x].w[y].w'' \mapsto_{red} w'w''$, applicable if $w[x] = o^-, w[y] = o^+$ for some oIo.

(R3) $w'.w[x].w[y].w'' \mapsto_{red} w'w[y]w[x]w''$, applicable if $w[x]Iw[y]$.

Reducing a word u to a word v using these rules is denoted by $u \mapsto_{red}^* v$. A reduction of $w = a_1 \dots a_n \in X_\Gamma^*$ to ε is the same as the free reduction of the sequence $a_1, a_2, \dots, a_n$. For any computation $w \in X_\Gamma^*$, we have $w \equiv_\Gamma \varepsilon$ iff w admits a reduction to ε [29, Equation (8.2)].

Assume that $\pi = w \mapsto_{red}^* \varepsilon$ is a reduction that transforms w into ε. The relation R_π relates positions of w which cancel in π:

$$w[x]R_\pi w[y] \ \text{ if } \ w'.w[x].w[y].w'' \mapsto_{red} w'.w'' \ \text{ or } \ w'.w[y].w[x].w'' \mapsto_{red} w'.w'' \ \text{ is used in } \ \pi$$

Greedy reductions. A word $w \in X_\Gamma^*$ is called *irreducible* if neither of the rules **R1** and **R2** is applicable in w. A reduction $\pi\colon w \mapsto_{red}^* \varepsilon$ is called *greedy* if it begins with a sequence of applications of **R1** and **R2** for each context so that the resulting context is irreducible. Note that every word w with $w \equiv_\Gamma \varepsilon$ has a greedy reduction: One can first (greedily) apply **R1** and **R2** until each context is irreducible. Since the resulting word w' still satisfies $w' \equiv_\Gamma \varepsilon$, there exists a reduction $w' \mapsto_{red}^* \varepsilon$. In total, this yields a greedy reduction.

Weak dependence. In the case of $\Gamma = \mathsf{MP}_{r,s}$, we know that any two vertices u, v are either dependent (i.e. belong to the same pushdown) or Γ is the direct product of graphs Γ_u and Γ_v such that u belongs to Γ_u and v belongs to Γ_v. This means, two operations that are not dependent can, inside every computation, be moved past each other without changing the effect on the stacks. This is not the case in general graphs. In Γ_2 in Figure 1, the vertices a and b are not dependent, but in the computation acb, they cannot be moved past each other, because none of them commutes with c. We therefore need the additional notion of weak dependence. We say that two vertices $u, v \in V$ are *weakly dependent* if there is a path between them in the complement of the graph. Here, the *complement* of a graph $\Gamma = (V, I)$ is obtained by complementing the independence relation ($v_1 I v_2$ in Γ iff we do not have $v_1 I v_2$ in the complement of Γ). Equivalently, u and v are not weakly dependent if Γ is the direct product of graphs Γ_u and Γ_v such that u belongs to Γ_u and v belongs to Γ_v. As observed above, Γ_2 shows that in general, weakly dependent vertices need not be dependent.

It can be seen that weak dependence is an equivalence relation on the set of vertices V, where the equivalence classes are the connected components in the complement of Γ. Note that all operations inside a context must belong to the same weak dependence class. We therefore say that two contexts c_1, c_2 are *weakly dependent* if their operations belong to the same weakly dependent equivalence class. Equivalently, two contexts are weakly dependent if all their letters are pairwise weakly dependent. In particular, weak dependency is an equivalence relation on contexts also. Let us denote the weak dependence equivalence relation by $\sim_W$ and by $[\ ]_{\sim_W}$ the set of all equivalence classes induced by $\sim_W$.

Scope bounded runs. We now define the notion of bounded scope computations. We first phrase the classical notion[1] of scope-boundedness [25] in our framework. If $\Gamma = \mathsf{MP}_{r,s}$, then $w \in X_\Gamma^*$ is considered k-scope bounded if there is a reduction π for w such that in between any two symbols $w[i]$ and $w[j]$ related in R_π, at most k contexts visit the same anti-clique of $w[i]$ and $w[j]$. Note that in $\mathsf{MP}_{r,s}$, for every reduction, there is a greedy reduction that induces the same relation R_π. Indeed, any applications of **R1** and **R2** that are applicable in a context at the start will eventually be made anyway: In $\mathsf{MP}_{r,s}$, if a word reduces to ε, then every position has a uniquely determined "partner position" with which is cancels in every possible reduction. Therefore, we generalize scope boundedness as follows[2].

▶ **Definition 3.1** (Scope Bounded Computations). *Consider a computation $w \in X_\Gamma^+$. We say w is k-scoped if there is a greedy reduction $\pi = w \mapsto_{red}^* \varepsilon$ such that in between any two symbols $w[i]$ and $w[j]$ related by R_π, at most $k-1$ contexts between $w[i]$ and $w[j]$ belong to the same weak dependence class as $w[i]$ and $w[j]$.*

By $\mathsf{sc}(w)$, we denote the smallest number k so that w is k-scoped. Note that there is such a k if and only if $w \equiv_\Gamma \varepsilon$. Thus, if $w \not\equiv_\Gamma \varepsilon$, we set $\mathsf{sc}(w) = \infty$. In the example in Figure 1 (graph Γ_1) the computation $w = b_1^+(c_2^+a^+c_1^+a^+c_1^-a^-c_2^-a^-)^m b_1^-$ is 3-scoped for all values of m, even though the number of context switches grows with m.

Interaction distance. We make the notion of scope bound more formal using the notion of interaction distance. Given a computation $w \in X_\Gamma^+$. Let $c_1c_2\ldots c_n$ be the canonical decomposition of w into contexts. We say that two contexts c_i, c_j with $i < j$ have an *interaction distance* K if there are $K-1$ contexts between c_i and c_j which are weakly dependent with c_i. Consider the computation $b_1^+(a^+c_1^+)^{m_1}b_2^+(a^+c_2^+)^{m_2}b_3^+(a^+c_3^+)^{m_3}b_3^-(c_3^-a^-)^{m_3}b_2^-(c_2^-a^-)^{m_2}b_1^-(c_1^-a^-)^{m_1}$. Each differently colored sequence is a context. The interaction distance between b_1^+ and b_1^- is 5, since the weakly dependent contexts strictly between them are $b_2^+, b_3^+, b_3^-, b_2^-$.

[1] The conference version [22] contains a slightly more restrictive definition. We follow the journal version [25].

[2] Another natural notion of scope-boundedness can be obtained by dropping the greediness condition. Hence, we would ask for a reduction π such that between any two R_π-related positions, there are at most $k-1$ contexts in the same weak dependence class. We expect that with this notion, Theorems 4.1 and 4.3 would still hold, but this would require changes to the algorithms.

Thus, w is k-scoped if and only if there is a greedy reduction $\pi\colon w \mapsto_{red} \varepsilon$ such that whenever $w[i]R_\pi w[j]$, then the contexts of $w[i]$ and $w[j]$ have interaction distance at most k.

The following is the central decision problem studied in this paper.

The Bounded Scope Reachability Problem(BSREACH)
Given: Graph Γ, scope bound k, valence system $\mathcal{A}$ over Γ, initial state q_{init}, final state q_{fin}
Decide: Is there a run from (q_{init}, ε) to (q_{fin}, w), for some $w \in X_\Gamma^*$ with $\mathsf{sc}(w) \le k$?

Thus, in BSREACH, both Γ and k are part of the input. We also consider versions where certain parameters are fixed: If Γ is fixed, we denote the problem by BSREACH(Γ). If Γ is part of the input, but can be drawn from a class $\mathcal{G}$ of graphs, we write BSREACH($\mathcal{G}$). Finally, if we fix k, we use a subscript k, resulting in the problems BSREACH$_k$, BSREACH$_k$(Γ), BSREACH$_k$($\mathcal{G}$).

Deciding whether there is a run (q_{init}, ε) to (q_{fin}, w) with $w \equiv_\Gamma \varepsilon$ corresponds to general configuration reachability [28]. Hence, we consider the scope-bounded version of configuration reachability.

Strongly Induced Subgraphs. When we study decision problems for valence systems over graph monoids, then typically, if Δ is an induced subgraph of Γ, then a problem instance for Δ can trivially be reduced to an instance over Γ. Here, induced subgraph means that Δ can be embedded into Γ so that there is an edge in Δ iff there is one in Γ.

This is not necessarily the case for BSREACH: An induced subgraph might decompose into different weak dependence classes than Γ. Therefore, we use a stronger notion of embedding. We say that $\Gamma' = (V', I')$ is a *strongly induced subgraph* of $\Gamma = (V, I)$ if there is an injective map $\iota\colon V \to V'$ such that for any $u, v \in V$, we have (i) uIv iff $\iota(u)I'\iota(v)$ and (ii) $u \sim_W v$ iff $\iota(u) \sim_W \iota(v)$. For example, the graph Γ consisting of two adjacent vertices (without loops) is an induced subgraph of Γ_2 in Figure 1. However, Γ is not a strongly induced subgraph of Γ_2: In Γ_2, a and b are weakly dependent, whereas the vertices of Γ are not.

Neighbor Antichains. Let $\Gamma = (V, I)$ be a graph. In our algorithms, we will need to store information about a dependent set $U \subseteq V$ from which we can conclude whether for another dependent set $U' \subseteq V$, we have UIU'; that is, for all $u \in U, u' \in U', uIu'$. To estimate the required information, we use the notion of neighbor antichains. Let $\Gamma = (V, I)$ be a graph. Given $v \in V$, let $N(v)$ represent the neighbors of v, that is $N(v) = \{u \in V \mid uIv\}$. We define a quasi-ordering on V as follows. For $u, v \in V$, we have $u \le v$ if $N(u) \subseteq N(v)$. It is possible that for distinct, $u, v \in V$ we have $u \le v$ and $v \le u$ and thus $\le$ is not necessarily a partial order. In the following, we will assume that the graphs Γ are always equipped with some linear order $\ll$ on V. For example, one can just take the order in which the vertices appear in a description of Γ. Using $\ll$, we can turn $\le$ into a partial order, which is easier to use algorithmically: We set $u \preceq v$ if and only if $u \le v$ and $u \ll v$.

Now, given $U \subseteq V$, let $\min U = \{u \in U \mid \forall v \in U \setminus \{u\}, v \not\preceq u\}$ and $\max U = \{u \in U \mid \forall v \in U \setminus \{u\}, u \not\preceq v\}$ denote the minimal and maximal elements of U, respectively.

▶ **Lemma 3.2.** *For sets $U, U' \subseteq V$, UIU' if and only if* $(\min U)I(\min U')$.

Since $\min U$ and $\min U'$ are antichains w.r.t. $\preceq$, if we bound the size of such antichains in our graph Γ, we bound the amount of information needed to store to determine whether UIU'. We call a subset $A \subseteq V$ a *neighbor antichain* if (i) A is dependent (i.e. an anti-clique, no edges between any two vertices of A) and (ii) A is an antichain with respect to $\preceq$. For the graph Γ_1 in Figure 1, each vertex is a neighbor antichain, while for $\overline{\Gamma}_1$, each $\{a, b_i, c_j\}$ is a neighbor antichain for all i, j. By $\tau(\Gamma)$, we denote the maximal size of a neighbor antichain in Γ. Thus $\tau(\Gamma_1) = 1, \tau(\overline{\Gamma}_1) = 3$. We say that a class $\mathcal{G}$ of graphs is *neighbor antichain bounded* if there is a number t such that $\tau(\Gamma) \leq t$ for every graph Γ in $\mathcal{G}$.

For example, the class of graphs $\mathcal{G}$ consisting of bipartite graphs B_n with nodes $\{u_i, v_i \mid i \in \{1, \ldots, n\}\}$, where $\{u_i, v_j\}$ is an edge iff $i \neq j$, is not neighbor antichain bounded.

4 Main results

In this section, we present the main results of this work. If both the graph and the scope bound k are part of the input, the bounded scope reachability problem is PSPACE-complete (as we will show in Theorem 4.1). Since graph monoids provide a much richer class of storage mechanisms than multi-pushdowns, this raises the question of how the complexity is affected if the storage mechanism (i.e. the graph) is drawn from a subclass of all graphs.

▶ **Theorem 4.1** (Scope bound in input). *Let $\mathcal{G}$ be a class of graphs. Then* BSREACH($\mathcal{G}$) *is*
1. NL-*complete if the graphs in $\mathcal{G}$ have at most one vertex,*
2. P-*complete if every graph in $\mathcal{G}$ is an anti-clique and $\mathcal{G}$ contains a graph with ≥ 2 vertices,*
3. PSPACE-*complete otherwise.*

▶ **Corollary 4.2.** *Let Γ be a graph. Then* BSREACH(Γ) *is*
1. NL-*complete if Γ has at most one vertex,*
2. P-*complete if Γ is an anti-clique with ≥ 2 vertices,*
3. PSPACE-*complete otherwise.*

Fixed scope bound. We notice that the problem BSREACH($\mathcal{G}$) is below PSPACE only for severely restricted classes $\mathcal{G}$, where bounded scope reachability degenerates into ordinary reachability in pushdown automata or one-counter automata. Therefore, we also study the setting where the scope bound k is fixed. However, our result requires two assumptions on the graph class $\mathcal{G}$. The first assumption is that $\mathcal{G}$ be closed under taking strongly induced subgraphs. This just rules out pathological exceptions: otherwise, it could be that there are hard instances for BSREACH_k that only occur embedded in extremely large graphs in $\mathcal{G}$, resulting in lower complexity. In other words, we restrict our attention to the cases where an algorithm for $\mathcal{G}$ also has to work for strongly induced subgraphs. For each individual graph, this is always the case: if Δ is a strongly induced subgraph of Γ, then $\text{BSREACH}_k(\Delta)$ trivially reduces to $\text{BSREACH}_k(\Gamma)$.

Our second assumption is that $\mathcal{G}$ be neighbor antichain bounded. This is a non-trivial assumption that still covers many interesting types of infinite-state systems from the literature. For example, every graph mentioned in Example 2.1 has neighbor antichains of size at most 1. In particular, our result still generalizes the case of multi-pushdown systems.

Moreover, consider the graphs SC_m for $m \in \mathbb{N}$, where (i) SC_0 is a single unlooped vertex, (ii) SC_{2m+1} is obtained from SC_{2m} by adding a new vertex adjacent to all existing vertices, and (iii) SC_{2m+2} is obtained from SC_{2m+1} by adding an isolated unlooped vertex. Then neighbor antichains in SC_m are of size at most 1. Furthermore, using reductions from [31, Proposition 3.6], it follows that whenever reachability for valence systems over Γ is decidable, then this problem reduces in polynomial time to reachability over some SC_m. Whether reachability is decidable for the graphs SC_m remains an open problem [31]. Thus the graphs SC_m form an extremely expressive class that is still neighbor antichain bounded.

▶ **Theorem 4.3** (Fixed scope bound). *Let $\mathcal{G}$ be closed under strongly induced subgraphs and neighbor antichain bounded. For every $k \geq 1$, the problem* $\mathsf{BSREACH}_k(\mathcal{G})$ *is*

1. NL*-complete if $\mathcal{G}$ consists of cliques of bounded size,*
2. P*-complete if $\mathcal{G}$ contains some graph that is not a clique, and the size of cliques in $\mathcal{G}$ is bounded,*
3. PSPACE*-complete otherwise.*

In Theorem 4.3, we do not know if one can lift the restriction of neighbor antichain boundedness. In Section 8, we describe a class of graphs that is closed under strongly induced subgraphs, but we do not know the exact complexity of $\mathsf{BSREACH}_k(\mathcal{G})$.

Theorem 4.3 allows us to deduce the complexity of $\mathsf{BSREACH}_k(\Gamma)$ for every Γ.

▶ **Corollary 4.4.** *Let Γ be a graph. Then for every $k \geq 1$, the problem* $\mathsf{BSREACH}_k(\Gamma)$ *is*

1. NL*-complete if Γ is a clique,*
2. P*-complete otherwise.*

Proof. Apply Theorem 4.3 to the class consisting of Γ and its strongly induced subgraphs. ◀

Discussion of results. In the case of multi-pushdown systems, La Torre, Napoli, and Parlato [25] show that scope-bounded reachability belongs to PSPACE, and is PSPACE-hard if either the number of stacks or the scope bound k is part of the input. Our results complete the picture in several ways. If k is part of the input, then PSPACE-hardness even holds if we have two $\mathbb{N}$-valued counters instead of stacks (Theorem 4.1). Moreover, hardness also holds when we have two $\mathbb{Z}$-valued counters (which often exhibit lower complexities [14]). Moreover, we determine the complexity the case that both k and the number s of stacks is fixed.

Our results can also be interpreted in terms of vector addition systems with states (VASS). In the case of VASS (i.e. unlooped cliques), our results imply that scope-bounded reachability is PSPACE-complete if either (i) the number d of counters or (ii) the scope-bound k are part of the input (and $d \geq 2$). The same is true if we have integer VASS [14] (looped cliques).

Thus, for VASS, scope-bounding reduces the complexity of reachability from at least non-elementary [8] to PSPACE. Interestingly, for two counters, the complexity goes up from NL for general reachability [11] to PSPACE. For integer VASS, we go up from NP for general reachability [14] and for a fixed number of counters even from NL [13], to PSPACE.

Note that we obtain a much more complete picture compared to what is known for bounded context switching [19]. There, even the complexity for many individual graphs is not known. Moreover, the case of fixed context bounds has not been studied in the case of bounded context switching.

5 Block decompositions

In this section, we lay the foundation for our decision procedure in Section 6. We show that in every scope-bounded run w, each context can be decomposed into a bounded number of "blocks", which will guarantee that w can be reduced to ε by way of "block-wise" reductions. In our algorithms, this will allow us to abstract from each block (which can have unbounded length) by a finite amount of data. This is similar to the block decomposition in [19].

Let $w \in X_\Gamma^*$ such that $w \equiv_\Gamma \varepsilon$ with a reduction π. We call a decomposition $w = w_1 \cdots w_m$ a *block decomposition* if it refines the canonical context decomposition[3] and for each w_i, there is a w_j such that R_π relates every position in w_i with a position in w_i itself or in w_j. Here, we do not rule out $i = j$: A block may itself reduce to ε.

[3] In other words, each context in w consists of a contiguous subset of the factors $w_1, \ldots, w_m$.

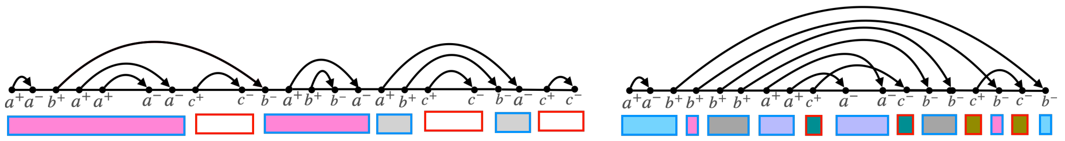
$a^+ a^- b^+ a^+ a^+ a^- a^- c^+ c^- b^- a^+ b^+ b^- a^- a^+ b^+ c^+ c^- b^- a^- c^+ c^-$
$a^+ a^- b^+ b^+ b^+ b^+ a^+ a^+ c^+ a^- a^- c^- b^- b^- c^+ b^- c^- b^-$

Figure 2 Context and block decomposition for a computation over $\overline{\Gamma}_2$ in Figure 1. The blue and red lined rectangles are the two contexts. The color filled rectangles represent blocks, with partner cancelling blocks having the same color filling.

Free reductions. Block decompositions are closely related to free reductions. Let $w_1, \ldots, w_m$ be a sequence of computations in X_Γ^*. A *free reduction* is a finite sequence of applications of the rewriting rules below to consecutive entries of the sequence so that $w_1, \ldots, w_m$ gets transformed into the empty sequence.

(FR1) $w_i, w_j \mapsto_{free} \varepsilon$ if $w_i w_j \equiv_\Gamma \varepsilon$
(FR2) $w_i, w_j \mapsto_{free} w_j, w_i$ if $w_i I w_j$
(FR3) $w_i \mapsto_{free} \hat{w}_i$ if $w_i \overset{*}{\mapsto}_{red} \hat{w}_i$ using rules **R1** and **R2**

We say that $w_1, \ldots, w_n$ is *freely reducible* if it admits a free reduction to the empty sequence.

As in [19], we have:

▶ **Proposition 5.1.** *If the decomposition $w = w_1 \cdots w_m$ refines the context decomposition, then it is a block decomposition if and only if the sequence $w_1, \ldots, w_m$ is freely reducible.*

The main result of this section is that in a scope-bounded run, there exists a block decomposition with a bounded number of factors in each context.

▶ **Theorem 5.2.** *Let $w \in X_\Gamma^*$ with $\mathsf{sc}(w) \leq k$. Then, there exists a block decomposition of w such that each context splits into at most $2k$ blocks.*

Let us sketch the proof. The block decomposition is obtained by scanning each context c from left to right. As long as there is another context c' such that all symbols either cancel inside c or with a symbol in c', we add symbols to the current block. When we encounter a symbol that cancels with a position outside of c and c', we start a new block. We show that this yields a block decomposition (see Figure 2 for an example) and with arguments similar to [19], one can show that it results in at most $2k$ blocks per context.

6 Decision procedure

In this section, we present the algorithms for the upper bounds of Theorems 4.1 and 4.3.

Block abstractions. The algorithm for bounded context switching in [19] abstracts each block by a non-deterministic automaton. This approach uses polynomial space per block, which would not be a problem for our PSPACE algorithm. However, for our P and NL algorithms, this would require too much space. Therefore, we begin with a new notion of "block abstraction", which is more space efficient.

Let $w = w_1 \cdots w_m$ be a block decomposition for a run of a valence system $\mathcal{A}$. Then it follows from Proposition 5.1 that there are words $\hat{w}_1, \ldots, \hat{w}_m$ such that $w_i \overset{*}{\mapsto}_{red} \hat{w}_i$ for each i such that the sequence $\hat{w}_1, \ldots, \hat{w}_m$ can be reduced to the empty sequence using the rewriting rules **FR1** and **FR2**. For each i, we store (i) the states occupied at the beginning and end of w_i, (ii) its first operation $f \in X_\Gamma$ in w_i, (iii) a non-deterministically chosen operation $o \in X_\Gamma$ occurring in w_i, and (iv) sets U_i^{min} and U_i^{max}, such that every maximal vertex occurring in w_i

is contained in U_i^{max} and every minimal vertex occurring in $\hat{w}_i$ is contained in U_i^{min}. In this case, we will say that the block abstraction "represents" the word $\hat{w}_i$. Thus, a *block abstration* is a tuple $N = (q_1, q_2, f, o, U^{\mathsf{min}}, U^{\mathsf{max}})$, where q_1, q_2 are states in $\mathcal{A}$, $f, o \in X_\Gamma$ are symbols, and $U^{\mathsf{min}}, U^{\mathsf{max}} \subseteq V$ are neighbor antichains. Note that for every set $B \subseteq V$, the sets $\mathsf{min}\, B$ and $\mathsf{max}\, B$ are neighbor antichains. Formally, we say that $N = (q_1, q_2, f, o, U^{\mathsf{min}}, U^{\mathsf{max}})$ *represents* $\hat{u} \in X_\Gamma^*$ if there is a word $u \in X_\Gamma^*$ such that (i) $u \mapsto_{red}^* \hat{u}$, (ii) u is read on some path from q_1 to q_2, (iii) u begins with f, (iv) o occurs in u, (iv) the set of vertices B occurring in u is a dependent set, (v) we have $\mathsf{max}\, B \subseteq U^{\mathsf{max}}$, and (vi) $\mathsf{min}\, \hat{B} \subseteq U^{\mathsf{min}}$, where $\hat{B}$ is the set of vertices occurring in $\hat{u}$. Then, $L(N)$ denotes the set of all words represented by N. We say that two block abstractions $N = (q_1, q_2, f, o, U^{\mathsf{min}}, U^{\mathsf{max}})$ and $N' = (q_1', q_2', f', o', U'^{\mathsf{min}}, U'^{\mathsf{max}})$ are *dependent* if $U^{\mathsf{max}} \cup U'^{\mathsf{max}}$ is a dependent set.

Context abstractions. Similarly, we will also need to abstract contexts. For this, we need to abstract each of its blocks. In addition, we need to store the whole context's first symbol (f) and some non-deterministically chosen other symbol (o). These additional symbols will be used to verify that we correctly guessed the canonical context decomposition of w. Thus, a *context abstraction* is a tuple $\mathcal{C} = (N_1, \ldots, N_{2k}, f, o)$, where $N_1, \ldots, N_{2k}$ are pairwise dependent block abstractions, and f and o are symbols. We say that a context abstraction $\mathcal{C} = (N_1, \ldots, N_{2k}, f, o)$ is *independent* with a context abstraction $\mathcal{C}' = (N_1', \ldots, N_{2k}', f', o')$ if (i) $f' I o$ and (ii) for some $i \in \{1, \ldots, 2k\}$, the block abstraction $N_i = (q_1^i, q_2^i, f_i, o_i, U_i^{\mathsf{min}}, U_i^{\mathsf{max}})$ satisfies $o = o_i$. Intuitively, this means if we have a word represented by $\mathcal{C}'$ and then append a word represented by $\mathcal{C}$, then these words will be the contexts in the canonical context decomposition.

In our algorithms, we will need to check whether a block decomposition admits a free reduction. This means, we need to check whether the words represented by block abstractions can cancel (to apply rule FR1) or commute (to apply FR2). Let us see how to do this. We say that the block abstractions $N = (q_1, q_2, f, o, U^{\mathsf{min}}, U^{\mathsf{max}})$ and $N' = (q_1', q_2', f', o', U'^{\mathsf{min}}, U'^{\mathsf{max}})$ *commute* if $U^{\mathsf{min}} I U'^{\mathsf{min}}$. Note that if N and N' commute, then uIu' for every $u \in L(N)$ and $u' \in L(N')$. We need an analogous condition for cancellation. We say that N and N' *cancel* if there are words $u \in L(N)$ and $u' \in L(N')$ such that $uu' \equiv_\Gamma \varepsilon$. This allows us to define an analogue of free reductions on block abstractions.

▶ **Definition 6.1.** *A* free reduction *on a sequence $N_1, \ldots, N_m$ of block abstractions is a sequence of operations*
(FRA1) *$N_i, N_j \to_{free} \varepsilon$, if N_i and N_j cancel*
(FRA2) *$N_i, N_j \to_{free} N_j, N_i$, if N_i and N_j commute.*

Together with Lemma 3.2, the following lemma allows us to verify the steps in a free reduction on block abstractions.

▶ **Lemma 6.2.** *Given Γ, a valence system over Γ, and block abstractions N_1 and N_2, one can decide in* P *whether N_1 and N_2 cancel. If Γ is a clique, this can be decided in* NL*.*

Given block abstractions N_1 and N_2, (i) first perform saturation [19], obtaining irreducible blocks. Saturation can be implemented using reachability in a one counter automaton, known to be NL-complete [9], (ii) second, construct a pushdown automaton that is non-empty if and only if the saturated N_1 and N_2 cancel. Emptiness of pushdown automata is decidable in P. If Γ is a clique, then the pushdown automaton uses only a single stack symbol. Hence, we only need a one-counter automaton, for which emptiness is decidable in NL [9]. This yields the two upper bounds claimed in Lemma 6.2.

Reduction to a Reachability Graph. In all our algorithms, we reduce BSREACH for a valence system $\mathcal{A}$ over Γ to reachability in a finite graph $\mathcal{R}$. For the PSPACE algorithm, we argue that each node of $\mathcal{R}$ requires polynomially many bits and the edge relation can be decided in PSPACE. For the P and the NL algorithm, we argue that $\mathcal{R}$ is polynomial in size. Moreover, for the P algorithm, we compute $\mathcal{R}$ in polynomial time. For the NL algorithm, we show that the edge relation of $\mathcal{R}$ can be decided in NL.

The vertices of $\mathcal{R}$ maintain k context abstractions (hence $2k^2$ block abstractions) per weak dependence class. Let $d \leq |V|$ be the number of weak dependence classes. The idea behind this choice of vertices is to build up a computation w from left to right, by guessing k context abstractions $(\mathcal{C}_1^\gamma, \mathcal{C}_2^\gamma, \cdots, \mathcal{C}_k^\gamma)$ per equivalence class γ which forms the "current window" of w. The initial vertex consists of k tuples $(\mathcal{E}, \dots, \mathcal{E})$ per weak dependence equivalence class, where $\mathcal{E}$ is a placeholder representing a cancelled block or an empty block. An edge is added from a vertex v_1 to a vertex v_2 in the graph when the left most context in v_1 corresponding to an equivalence class γ cancels out completely, and v_2 is obtained by appending a fresh context abstraction $\mathcal{C}_y^\gamma$ to v_1. Indeed if w is k scope bounded, then the blocks of the first context abstraction $\mathcal{C}_1^\gamma$ must cancel out with blocks from the remaining context abstractions $\mathcal{C}_2^\gamma, \cdots, \mathcal{C}_k^\gamma$ using the free reduction rules discussed above. We can guess an equivalence class γ whose context abstraction $\mathcal{C}_1^\gamma$ cancels out, and extend w by guessing the next context abstraction $\mathcal{C}_y^\gamma$ in the same equivalence class. An edge between two vertices in $\mathcal{R}$ represents an extension of w where a new context of an appropriate equivalence class is added, after the leftmost context has cancelled out using some free reduction rules.

For a weak dependence class, we refer to the tuple of k contexts of interest as a *configuration*. Thus, a vertex in $\mathcal{R}$ consists of d configurations. As discussed earlier in section 6, we represent the $2k^2$ blocks in each configuration using block abstractions.

▶ **Definition 6.3.** *Given a weak dependence class $\gamma \in [\,]_{\sim_W}$, a* configuration of γ *is a k-tuple of the form $s_\gamma = (\mathcal{C}_1^\gamma, \mathcal{C}_2^\gamma, \cdots, \mathcal{C}_k^\gamma)$, where each $\mathcal{C}_c^\gamma$ for $1 \leq c \leq k$ is a context abstraction.*

As mentioned above, in slight abuse of terminology, in case of cancellation in free reductions, we also allow $\mathcal{E}$ as a placeholder for cancelled contexts.

Intuitively, the configuration tracks the remaining non-cancelled blocks of the last k contexts of this weak dependence class along with their relative positions in their contexts.

▶ **Definition 6.4.** *A vertex in the graph $\mathcal{R}$ has the form $(s_{\gamma_1}, \dots, s_{\gamma_d} | i, q)$ where (i) $\gamma_1, \dots, \gamma_d$ are the distinct equivalence classes in $[\,]_{\sim_W}$, (ii) each s_{γ_j} is a configuration, (iii) $i \in \{1, \cdots, d\}$ is the current weak dependence class we are on, and (iv) q is the last state occurring in s_{γ_i}.*

Here, if s_{γ_i} consists just of $\mathcal{E}$, then the last condition is satisfied automatically.

▶ **Definition 6.5.** *For $\gamma \in [\,]_{\sim_W}$, a configuration s'_γ is* one-step reachable *from a configuration $s_\gamma = (\mathcal{C}_1^\gamma, \dots, \mathcal{C}_k^\gamma)$ iff there exists some context abstraction $\mathcal{C}$ and a sequence of free reduction operations on the sequence of block abstractions*

$$N_1^{\gamma 1}, \dots, N_{2k}^{\gamma 1}, N_1^{\gamma 2}, \dots, N_{2k}^{\gamma 2}, \dots N_1^{\gamma k}, \dots, N_{2k}^{\gamma k}, N_1, \dots, N_{2k}$$

resulting in the new sequence (placing $\mathcal{E}$ in a position if the block was cancelled due to the free reduction rules; otherwise we keep the same block abstraction)

$$N_1'^{\gamma 1}, \dots, N_{2k}'^{\gamma 1}, N_1'^{\gamma 2}, \dots, N_{2k}'^{\gamma 2}, \dots N_1'^{\gamma k}, \dots, N_{2k}'^{\gamma k}, N_1'^{\gamma (k+1)}, \dots, N_{2k}'^{\gamma (k+1)}$$

such that $N_\ell'^{\gamma 1} = \mathcal{E}$, for all $\ell \in \{1, \dots, 2k\}$, and $\left(\mathcal{C'}_2^\gamma, \mathcal{C'}_3^\gamma, \cdots, \mathcal{C'}_k^\gamma, \mathcal{C'}_{k+1}^\gamma\right) = s'_\gamma$, where $\mathcal{C'}_\ell^\gamma = (N_1'^{\gamma \ell}, N_2'^{\gamma \ell}, \cdots, N_{2k}'^{\gamma \ell}, f^{\gamma \ell}, o^{\gamma \ell})$ for $\ell \in \{1, \dots, k+1\}$, and $N_1, \dots, N_{2k}$ are the block abstractions in $\mathcal{C}$. In this case, we write $s_\gamma \xrightarrow{\mathcal{C}} s'_\gamma$.

In short, we can go from s_γ to s'_γ in one step if we can add some context abstraction to s_γ so that using free reduction steps, we can reach s'_γ along with clearing the oldest context.

We are now ready to define the edge relation of $\mathcal{R}$.

▶ **Definition 6.6.** *There is an edge in $\mathcal{R}$ from a vertex $v = (s_{\gamma_1}, \ldots, s_{\gamma_d} | j, q)$ to a vertex v' iff there is some $i \in \{1, \ldots, d\}$ and a configuration s'_{γ_i} such that (i) $s_{\gamma_i} \xrightarrow{\mathcal{C}'} s'_{\gamma_i}$ for some context abstraction $\mathcal{C}'$ such that $\mathcal{C}'$ is independent with the last context abstraction $\mathcal{C}$ of s_{γ_j} and (ii) q is the first state in $\mathcal{C}'$ and (iii) $v' = (s_{\gamma_1}, \ldots, s_{\gamma_{i-1}}, s'_{\gamma_i}, s_{\gamma_{i+1}}, \ldots, s_{\gamma_d} | i, q')$, where q' is the last state of $\mathcal{C}'$.*

Since a context is a maximal dependent sequence, this check suffices to guarantee independence of words represented by $\mathcal{C}'$ and $\mathcal{C}$.

As mentioned above, our algorithm reduces scope-bounded reachability to reachability between two nodes in $\mathcal{R}$. Details can be found in the full version.

Complexity. We turn to the upper bounds in Theorems 4.1 and 4.3. Asymptotically, a block abstraction requires $\log|Q| + 2t \cdot \log|V|$ bits, where t is an upper bound on the size of neighbor antichains. Per context, we store $2k$ block abstractions and two symbols. Let d be the number of weak dependence classes. In a node of $\mathcal{R}$, we store k contexts per weak dependence class, a number in $\{1, \ldots, d\}$, and a state. Hence, asymptotically, we need $M = dk^2(\log|Q| + t \cdot \log|V|) + \log d + \log|Q|$ bits per node of $\mathcal{R}$. To simulate the free reduction, we only need a constant multiple of this. We can thus decide reachability in $\mathcal{R}$ in PSPACE.

▶ **Proposition 6.7.** BSREACH *is in* PSPACE.

We now look at the upper bounds for the first and second cases of Theorem 4.3.

▶ **Proposition 6.8.** *Let $\mathcal{G}$ be a class of graphs that is closed under strongly induced subgraphs and neighbor antichain bounded. If $\mathcal{G}$ consists of cliques of bounded size, then for each $k \geq 1$, the problem* BSREACH$_k(\mathcal{G})$ *belongs to* NL.

Proof. Our assumptions imply that $d \leq |V|$, t, and k are bounded. Thus M is at most logarithmic in the input. Moreover, we can simulate free reductions using logarithmic space, because checking whether two block abstractions cancel can be done in NL by Lemma 6.2. ◀

▶ **Proposition 6.9.** *Let $\mathcal{G}$ be a class of graphs that is closed under strongly induced subgraphs and neighbor antichain bounded. If the size of cliques in $\mathcal{G}$ is bounded, then for every $k \geq 1$, the problem* BSREACH$_k(\mathcal{G})$ *belongs to* P.

Proof. First observe that as in Proposition 6.8, the parameters d, t, and k are bounded. To see this for d, let ℓ be an upper bound on the size of cliques in $\mathcal{G}$. Then, every graph Γ in $\mathcal{G}$ can have at most ℓ weak dependence classes: Otherwise, Γ would have a clique with $\ell + 1$ nodes as a strongly induced subgraph, and thus $\mathcal{G}$ would contain a clique with $\ell + 1$ nodes. Hence, d is bounded and for a node in $\mathcal{R}$, we need only logarithmic space. Moreover, by Lemma 6.2, we can verify a free reduction step in P. ◀

Special Graphs. We turn to the NL and P upper bounds for Theorem 4.1. In each case, all graphs are anti-cliques. Thus, every run has a single context and BSREACH reduces to membership for pushdown automata, which is in P. If there is just one vertex, we can even obtain a one-counter automaton, for which emptiness is in NL [9].

▶ **Proposition 6.10.** *If $\mathcal{G}$ is the class of anti-cliques, then* BSREACH$(\mathcal{G})$ *is in* P. *Moreover, if Γ has only one vertex, then* BSREACH(Γ) *is in* NL.

7 Hardness

In this section, we show the hardness results of Theorems 4.1 and 4.3.

▶ **Proposition 7.1.** *If Γ is not a clique, then* $\mathsf{BSREACH}_k(\Gamma)$ *is* P*-hard for each $k \geq 1$.*

This uses standard techniques. If a valence system uses only the two non-adjacent vertices in Γ, then scope-bounded reachability is the same as ordinary reachability. If both vertices are looped, then this is the rational subset membership problem for a free group of rank 2, for which P-hardness was observed in [15, Theorem III.4]. If at least one vertex is unlooped, a standard encoding yields a reduction from emptiness of pushdown automata.

Two adjacent vertices. Our second hardness proof shows PSPACE-hardness in Theorem 4.1.

▶ **Proposition 7.2.** *If Γ has two adjacent nodes, then* $\mathsf{BSREACH}(\Gamma)$ *is* PSPACE*-hard.*

For the proof of Proposition 7.2, we employ the model of bounded queue automata. A *bounded queue automaton (BQA)* is a tuple $\mathcal{A} = (Q, n, T, q_0, q_f)$, where (i) Q is a finite set of *states*, (ii) $n \in \mathbb{N}$ is the *queue length*, given in unary, (iii) T is its set of *transitions*, (iv) $q_0 \in Q$ is its *initial state*, and (iv) $q_f \subseteq Q$ is its *final state*. A *configuration* of a BQA is a pair $(q, w) \in Q \times \{0,1\}^n$. A transition is of the form (q, x, y, q'), where $q, q' \in Q$ and $x, y \in \{0,1\}$. We write $(q, w) \to (q', w')$ if there is a transition (q, x, y, q') such that (i) w has prefix x and (ii) removing x from the left and appending y on the right yields w'. The *reachability problem for BQA* is the following: Given a bounded queue automaton (Q, n, T, q_0, q_f), is it true that $(q_0, 0^n) \xrightarrow{*} (q_f, 0^n)$? It is straightforward to simulate a linear bounded automaton using a bounded queue automaton and vice-versa, hence reachability for BQA is PSPACE-complete.

General idea and challenge. Let us first assume that the nodes u and v in Γ are not weakly dependent. The initial approach for Proposition 7.2 is to encode the queue content in the current window of $k = n$ contexts. In each context, we encode a 0-bit using an occurrence of the letter u^+ that can only be cancelled with a future u^-. We call this a 0*-context*. Likewise, a 1-bit is encoded by two occurrences of u, which we call a 1*-context*. Therefore, we abbreviate $\mathbf{0} = u^+$ and $\mathbf{1} = u^+u^+$. We also have the right inverses $\bar{\mathbf{0}} = u^-$ and $\bar{\mathbf{1}} = u^-u^-$. To start a new context, we multiply v^+v^- and use the abbreviation $\| = v^+v^-$. With this encoding, it is easy to check that the oldest context is a 0-context: Just multiply $\bar{\mathbf{0}} = u^-$ and then start a new context using $\| = v^+v^-$. This can only succeed if the oldest context encodes a 0: If it had encoded a 1, there would be another occurrence of u^+ that can never be cancelled.

However, it is not so easy to check that the oldest context is a 1-context. One could multiply with $\bar{\mathbf{1}}\| = u^-u^-v^+v^-$, but this can succeed even if the oldest context is a 0-context: Indeed, the first occurrence of u^- can cancel with the u^+ in the oldest context c, but the second u^- could cancel with u^+ in a context to the right of c.

Solution. We overcome this as follows. Instead of one context per bit, we use three contexts. To encode a 0 in the queue, we use a 0-context, a 1-context, and another 1-context, resulting in the string 011. To encode a 1, we do the same with the bit string 100. Then, we use the above approach to check for 011 or 100: Since a successful check for a 0-context guarantees that there was a 0-context, checking for $0, 1, 1$ guarantees that the oldest context is a 0-context, thus the three oldest contexts must carry 011. When we check for $1, 0, 0$, then among the three oldest contexts, at least two are 0-contexts, hence the three oldest contexts carry 100.

Let us calculate the required scope bound to implement this idea. Since we only use the operations u^+, u^-, v^+, v^-, we only have u-contexts (consisting of u^+, u^-) and v-contexts (consisting of v^+, v^-). When we read the oldest bit in the queue, we produce three new u-contexts (separated by v-contexts). Then, we need to write a new bit in the queue, which requires another three u-contexts (separated by v-contexts). The interaction distance to the oldest u-context that is part of the leftmost queue bit is always $k = 3(2n-1)$: The oldest bit is encoded using three u-contexts. For each further queue entry $i \in \{2, \ldots, n\}$, we have three u-contexts that were used to read an even older bit, and then three u-contexts that encode the i-th bit in the queue. In total, this yields $3(1 + 2(n-1)) = 3(2n-1)$ many u-contexts.

To initialize the queue, we use $t_0 = (\mathbf{0}\|\mathbf{1}\|\mathbf{1}\|)(\mathbf{e}\|\mathbf{e}\|\mathbf{e}\|\mathbf{0}\|\mathbf{1}\|\mathbf{1}\|)^{n-1}$. Here, $\mathbf{e} = u^+u^-$ is a "gap context" that ensures that the leftmost bit has interaction distance exactly $k = 3(2n-1)$ from the right end of t_0. Thus, t_0 puts n copies of the bit string 011, plus $n-1$ gap contexts into our window of $k = 3(2n-1)$ contexts. To simulate a transition (p, x, y, p'), we check that x is the bit encoded by the three oldest contexts. Afterwards, we put the new bit y into the queue. Therefore, if $x = 0$, define the triple $(x_1, x_2, x_3) = (\bar{\mathbf{0}}, \bar{\mathbf{1}}, \bar{\mathbf{1}})$; if $x = 1$, let $(x_1, x_2, x_3) = (\bar{\mathbf{1}}, \bar{\mathbf{0}}, \bar{\mathbf{0}})$. Moreover, if $y = 0$, then let $(y_1, y_2, y_3) = (\mathbf{0}, \mathbf{1}, \mathbf{1})$; if $y = 1$, then let $(y_1, y_2, y_3) = (\mathbf{1}, \mathbf{0}, \mathbf{0})$. Then we use the string $t_{x,y} = x_1\|x_2\|x_3\|y_1\|y_2\|y_3\|$. Finally, to check that the encoded queue content consists entirely of 0's, we use $t_f = (\bar{\mathbf{0}}\|\bar{\mathbf{1}}\|\bar{\mathbf{1}}\|)(\mathbf{e}\|\mathbf{e}\|\mathbf{e}\|\bar{\mathbf{0}}\|\bar{\mathbf{1}}\|\bar{\mathbf{1}}\|)^{n-1}$. With this encoding, it is straightforward to translate a BQA into a valence system over Γ.

Note that if u and v are weakly dependent, then the same construction works, except that we have to set $k = 6(2n-1)$, because now the v-contexts $\| = v^+v^-$ between two u-contexts count towards the interaction distance.

Unbounded cliques. We turn to PSPACE-hardness in Theorem 4.3.

▶ **Proposition 7.3.** *Suppose $\mathcal{G}$ be either the class of unlooped cliques or the class of looped cliques. Then for every $k \geq 1$, the problem* $\mathsf{BSREACH}_k(\mathcal{G})$ *is* PSPACE*-hard.*

Here it is convenient to reduce from bit vector automata, whose configuration consists of a state and a bit vector. In each step, they can read and modify one of the bits. A *bit vector automaton (BVA)* is a tuple (Q, n, T, q_0, q_f), where (i) Q is a finite set of states, (ii) n is the *vector length*, given in unary, (iii) a set T of *transitions*, (iv) $q_0 \in Q$ is its *initial state*, and (v) $q_f \in Q$ is its *final state*. A transition is of the form (p, i, x, y, q) with $p, q \in Q$, $i \in \{0, \ldots, n\}$, and $x, y \in \{0, 1\}$. It checks that i-th bit is currently x, and sets the i-th bit to y. Thus, a *configuration* of a bit vector automaton is a pair $(q, w) \in Q \times \{0,1\}^n$. By $\xrightarrow{*}$, we denote the reachability relation. The *reachability problem for BVA* asks, given a BVA (Q, n, T, q_0, q_f), is it true that $(q_0, 0^n) \xrightarrow{*} (q_f, 0^n)$? Again, a simulation of linear bounded automata is straightforward and this problem is PSPACE-complete.

Proof of Proposition 7.3. Let $\mathcal{A} = (Q, n, T, q_0, q_f)$ be a BVA. Moreover, depending on whether $\mathcal{G}$ is the class of looped or unlooped cliques, let $\Gamma = (V, I)$ be either a looped or an unlooped clique with $2n$ vertices, so let $V = \{a_i, b_i \mid i \in \{1, \ldots, n\}\}$. Our construction does not depend on whether Γ is looped or unlooped and we will show that it is correct in either case. We first illustrate the idea for maintaining a single bit using the vertices a_i, b_i. To ease notation, we now write v for v^+ and $\bar{v}$ for v^- when $v \in V$. Consider the string

$$w = (a_i^{r_1} b_i \bar{b}_i \bar{a}_i^{s_1} b_i \bar{b}_i)(a_i \bar{a}_i b_i \bar{b}_i)^k (a_i^{r_2} b_i \bar{b}_i \bar{a}_i^{s_2} b_i \bar{b}_i) \cdots (a_i \bar{a}_i b_i \bar{b}_i)^k (a_i^{r_m} b_i \bar{b}_i \bar{a}_i^{s_m} b_i \bar{b}_i).$$

Moreover, assume that for each $j = 1, \ldots, m$, we have $r_j, s_j \in \{k, 3k\}$. We think of $a_i^{r_j} b_i \bar{b}_i$ as an operation that stores 0 if $r_j = k$ and stores 1 if $r_j = 3k$. We think of $\bar{a}_i^{s_j} b_i \bar{b}_i$ as a read operation, where again $s_j = k$ stands for 0 and $s_j = 3k$ stands for 1. Here, the purpose of $b_i \bar{b}_i$ is to start a new context (since Γ is a clique). Moreover, each factor $(a_i \bar{a}_i b_i \bar{b}_i)^k$ produces

k contexts in the weak dependence class of a_i, where each context contains $a_i\bar{a}_i$. This means, each factor $(a_i\bar{a}_ib_i\bar{b}_i)^k$ enforces an interaction distance of $k+1$ between $\bar{a}_i^{s_j}$ and $a_i^{r_{j+1}}$, and thus prevents them from canceling with each other.

We claim that $\mathsf{sc}(w) \leq k$ if and only if each read operation reads the bit that was stored before. In other words, we have $\mathsf{sc}(w) \leq k$ if and only if $r_j = s_j$ for each $j \in \{1, \dots, m\}$. Moreover, this is true regardless of whether Γ is looped or unlooped. For the "if", note that each $a_i^{r_j}$ can cancel with $\bar{a}_i^{s_j}$ and in every other context, every letter (a_i, $\bar{a}_i$, b_i, $\bar{b}_i$) can cancel with its direct neighbor. Conversely, suppose $r_j \neq s_j$ for some j. If $r_j = 3k$ and $s_j = k$, then the context $a_i^{r_j} = a_i^{3k}$ sees only $3k-1$ occurrences of $\bar{a}_i$ in contexts at interaction distance $\leq k$: First, the context $\bar{a}_i^{s_i} = \bar{a}_i^k$ yields k occurrences. The other $2k-1$ contexts are of the form $a_i\bar{a}_i$ and each provides one occurrence of $\bar{a}_i$. In total, we have $k + 2k - 1 = 3k - 1$. It is thus impossible to cancel every letter in $a_i^{r_j}$. The case $r_j = k$ and $s_j = 3k$ is symmetric. This proves our claim. Using this encoding, it is now straightforward to simulate n bits. ◀

Propositions 7.1–7.3 complete our proofs: Theorem 4.1 follows from Propositions 7.1, 7.2, 6.7, and 6.10. Theorem 4.3 follows from Propositions 7.1, 7.3, and 6.7–6.9.

8 Conclusion

We have introduced a notion of scope-bounded reachability for valence systems over graph monoids. In the special case of graphs that correspond to multi-pushdowns, this notion coincides with the original notion of scope-boundedness introduced by La Torre, Napoli, and Parlato [25]. We have shown that with this notion, scope-bounded reachability is decidable in PSPACE, even if the graph and the scope bound k are part of the input.

In addition, we have studied the complexity of the problem under four types of restrictions: (i) k and the graph are part of the input, and the graph is drawn from some class $\mathcal{G}$ of graphs, (Theorem 4.1), (ii) k is part of the input and the graph is fixed (Corollary 4.2), (iii) k is fixed and the graph is drawn from some class $\mathcal{G}$ of graphs that is neighbor antichain bounded and closed under strongly induced subgraphs (Theorem 4.3) and (iv) k is fixed and the graph is fixed (Corollary 4.4). We have completely determined the complexity landscape in the situations (i)–(iv): In every case, we obtain NL-, P-, or PSPACE-completeness. These results settle the complexity of scope-bounded reachability for most types of infinite-state systems that fit in the framework of valence systems and have been considered in the literature.

Open Problem: Dropping neighbor antichain boundedness. We leave open what complexities can arise if in case (iii) above, we drop the assumption of neighbor antichain boundedness. In other words: k is fixed and the graph comes from a class $\mathcal{G}$ that is closed under strongly induced subgraphs. For all we know, it is possible that there are classes $\mathcal{G}$ for which the problem is neither NL-, nor P-, nor PSPACE-complete.

For example, for each $n \geq 0$, consider the bipartite graph B_n with nodes $\{u_i, v_i \mid i \in \{1, \dots, n\}\}$, where $\{u_i, v_j\}$ is an edge if and only if $i \neq j$. Moreover, let $\mathcal{G}$ be the class of graphs containing B_n for every $n \in \mathbb{N}$ and all strongly induced subgraphs. Observe that the cliques in $\mathcal{G}$ have size at most 2: B_n is bipartite and thus every clique in B_n has size at most 2. Moreover, the graphs B_n have neighbor antichains of unbounded size: The set $\{u_1, \dots, u_n\}$ is a neighbor antichain in B_n.

We currently do not know the exact complexity of $\mathsf{BSREACH}_k(\mathcal{G})$. By Theorem 4.3, the problem is P-hard and in PSPACE. Intuitively, our P upper bound does not apply because in each node of $\mathcal{R}$, one would have to remember n bits in order to keep enough information about commutation of blocks: For a subset $S \subseteq \{1, \dots, n\}$, let u_S be the product of all $u_1^+, \dots, u_n^+$, where we only include u_i^+ if $i \in S$. Then $u_S v_j^+ \equiv v_j^+ u_S$ if and only if $j \notin S$.

References

1 C. Aiswarya, Paul Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *Automated Technology for Verification and Analysis – 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014. doi:10.1007/978-3-319-11936-6_1.

2 S. Akshay, Paul Gastin, Shankara Narayanan Krishna, and Sparsa Roychowdhury. Revisiting underapproximate reachability for multipushdown systems. In *Tools and Algorithms for the Construction and Analysis of Systems – 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*, volume 12078 of *Lecture Notes in Computer Science*, pages 387–404. Springer, 2020. doi:10.1007/978-3-030-45190-5_21.

3 Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. Linear-time model-checking for multithreaded programs under scope-bounding. In *Automated Technology for Verification and Analysis – 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings*, volume 7561 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2012. doi:10.1007/978-3-642-33386-6_13.

4 Devendra Bhave, Shankara Narayanan Krishna, Ramchandra Phawade, and Ashutosh Trivedi. On timed scope-bounded context-sensitive languages. In *Developments in Language Theory – 23rd International Conference, DLT 2019, Warsaw, Poland, August 5-9, 2019, Proceedings*, volume 11647 of *Lecture Notes in Computer Science*, pages 168–181. Springer, 2019. doi:10.1007/978-3-030-24886-4_12.

5 P. Buckheister and Georg Zetzsche. Semilinearity and context-freeness of languages accepted by valence automata. In *Mathematical Foundations of Computer Science 2013 – 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2013. doi:10.1007/978-3-642-40313-2_22.

6 Aiswarya Cyriac. *Verification of communicating recursive programs via split-width. (Vérification de programmes récursifs et communicants via split-width)*. PhD thesis, École normale supérieure de Cachan, France, 2014. URL: https://tel.archives-ouvertes.fr/tel-01015561.

7 Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR 2012 – Concurrency Theory – 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 2012. doi:10.1007/978-3-642-32940-1_38.

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

9 Stéphane Demri and Régis Gascon. The effects of bounding syntactic resources on presburger LTL. In *Proceedings of TIME 2007*, pages 94–104. IEEE Computer Society, 2007. doi:10.1109/TIME.2007.63.

10 Emanuele D'Osualdo, Roland Meyer, and Georg Zetzsche. First-order logic with reachability for infinite-state systems. In *Proceedings of LICS 2016*, pages 457–466. ACM, 2016. doi:10.1145/2933575.2934552.

11 Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in two-dimensional unary vector addition systems with states is nl-complete. In *Proceedings of LICS 2016*, pages 477–484. ACM, 2016. doi:10.1145/2933575.2933577.

12 Javier Esparza, Pierre Ganty, and Tomás Poch. Pattern-based verification for multithreaded programs. *ACM Trans. Program. Lang. Syst.*, 36(3):9:1–9:29, 2014. doi:10.1145/2629644.

13 Eitan M. Gurari and Oscar H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Sciences*, 22(2):220–229, 1981. doi:10.1016/0022-0000(81)90028-3.

14 Christoph Haase and Simon Halfon. Integer vector addition systems with states. In *Proceedings of RP 2014*, volume 8762 of *Lecture Notes in Computer Science*, pages 112–124. Springer, 2014. doi:10.1007/978-3-319-11439-2_9.

15 Christoph Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. In *Proceedings of LICS 2019*, pages 1–14. IEEE, 2019. doi:10.1109/LICS.2019.8785850.

16 Serge Haddad and Denis Poitrenaud. Recursive Petri nets. *Acta Informatica*, 44(7):463–508, 2007.

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

18 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In *Proceedings of POPL 2011*, pages 283–294. ACM, 2011. doi:10.1145/1926385.1926419.

19 Roland Meyer, Sebastian Muskalla, and Georg Zetzsche. Bounded context switching for valence systems. In *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 12:1–12:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.12.

20 Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *Proceedings of TACAS 2005*, pages 93–107. Springer, 2005.

21 Ganesan Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Transactions on Programming languages and Systems (TOPLAS)*, 22(2):416–430, 2000.

22 Salvatore La Torre and Margherita Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR 2011 – Concurrency Theory – 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2011. doi:10.1007/978-3-642-23217-6_14.

23 Salvatore La Torre and Margherita Napoli. A temporal logic for multi-threaded programs. In *Theoretical Computer Science – 7th IFIP TC 1/WG 2.2 International Conference, TCS 2012, Amsterdam, The Netherlands, September 26-28, 2012. Proceedings*, volume 7604 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2012. doi:10.1007/978-3-642-33475-7_16.

24 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. Scope-bounded pushdown languages. *Int. J. Found. Comput. Sci.*, 27(2):215–234, 2016. doi:10.1142/S0129054116400074.

25 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. Reachability of scope-bounded multistack pushdown systems. *Inf. Comput.*, 275:104588, 2020. doi:10.1016/j.ic.2020.104588.

26 Salvatore La Torre and Gennaro Parlato. Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 173–184. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.173.

27 Georg Zetzsche. Silent transitions in automata with storage. In *Proceedings of ICALP 2013*, volume 7966 of *Lecture Notes in Computer Science*, pages 434–445. Springer, 2013. doi:10.1007/978-3-642-39212-2_39.

28 Georg Zetzsche. Computing downward closures for stacked counter automata. In *Proceedings of STACS 2015*, volume 30 of *LIPIcs*, pages 743–756. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.743.

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

30 Georg Zetzsche. Monoids as storage mechanisms. *Bull. EATCS*, 120, 2016. URL: `http://eatcs.org/beatcs/index.php/beatcs/article/view/459`.

31 Georg Zetzsche. The emptiness problem for valence automata over graph monoids. *Inf. Comput.*, 277:104583, 2021. `doi:10.1016/j.ic.2020.104583`.