THE EMPTINESS PROBLEM FOR VALENCE AUTOMATA OVER GRAPH MONOIDS

GEORG ZETZSCHE

Abstract. This work studies which storage mechanisms in automata permit decidability of the emptiness problem. The question is formalized using valence automata, an abstract model of automata in which the storage mechanism is given by a monoid. For each of a variety of storage mechanisms, one can choose a (typically infinite) monoid M such that valence automata over M are equivalent to (one-way) automata with this type of storage. In fact, many important storage mechanisms can be realized by monoids defined by finite graphs, called graph monoids. Examples include pushdown stacks, partially blind counters (which behave like Petri net places), blind counters (which may attain negative values), and combinations thereof.

Hence, we study for which graph monoids the emptiness problem for valence automata is decidable. A particular model realized by graph monoids is that of Petri nets with a pushdown stack. For these, decidability is a long-standing open question and we do not answer it here.

However, if one excludes subgraphs corresponding to this model, a characterization can be achieved. Moreover, we provide a description of those storage mechanisms for which decidability remains open. This leads to a model that naturally generalizes both pushdown Petri nets and the priority multicounter machines introduced by Reinhardt.

The cases that are proven decidable constitute a natural and apparently new extension of Petri nets with decidable reachability. It is finally shown that this model can be combined with another such extension by Atig and Ganty: We present a further decidability result that subsumes both of these Petri net extensions.

1. Introduction

For each storage mechanism in one-way automata, it is an important question whether the emptiness problem is decidable. It therefore seems prudent to aim for general insights into which properties of storage mechanisms are responsible for decidability or undecidability.

Our approach to obtain such insights is the model of valence automata. These feature a finite-state control and a (typically infinite) monoid that represents a storage mechanism. The edge inscriptions consist of an input word and an element of the monoid. Then, a computation is accepting if it arrives in a final state and composing the encountered monoid elements yields the neutral element. This way, by choosing a suitable monoid, one can realize a variety of storage mechanisms. Hence, our question becomes: *For which monoids M is the emptiness problem for valence automata over M decidable?*

We address this question for a class of monoids that was introduced in [19] and accommodates a number of storage mechanisms that have been studied in automata theory. Examples include *pushdown stacks*, *partially blind counters* (which behave like Petri net places), and *blind counters* (which may attain negative values; these are in most situations interchangeable with reversal-bounded counters), and combinations thereof. See [22, 23] for an overview. These monoids are defined by graphs and thus called *graph monoids*[1].

A particular type of storage mechanism that can be realized by graph monoids are partially blind counters that can be used simultaneously with a pushdown stack. Automata

The author is supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD) and by Labex DigiCosme, Univ. Paris-Saclay, project VERICONISS.

[1]They are not to be confused with the closely related, but different concept of *trace monoids* [5], i.e. monoids of Mazurkiewicz traces, which some authors also call graph monoids.

with such a storage are equivalent to *pushdown Petri nets (PPN)*, i.e. Petri nets where the transitions can also operate on a pushdown stack. This means, a complete characterization of graph monoids with a decidable emptiness problem would entail an answer to the long-standing open question of whether reachability is decidable for this Petri net extension [15]. Partial solutions have recently been obtained by Atig and Ganty [2] and by Leroux, Sutre, and Totzke [12].

Contribution. While this work does not answer this open question concerning PPN, it does provide a characterization among all graph monoids that avoid this elusive storage type. More precisely, we identify a set of graphs, 'PPN-graphs', each of which corresponds precisely to PPN with one Petri net place. Then, among all graphs Γ avoiding PPN-graphs as induced subgraphs, we characterize those for which the graph monoid $\mathbb{M}\Gamma$ results in a decidable emptiness problem. Furthermore, we provide a simple, more mechanical (as opposed to algebraic) description of

(i) the storage mechanism emerging as the most general decidable case and
(ii) a type of mechanism equivalent to the cases we leave open.

The model (i) is a new extension of partially blind counter automata (i.e. Petri nets). While the decidability proof employs a reduction to Reinhardt's priority multicounter machines [15], the model (i) seems to be expressively incomparable to Reinhardt's model. The model (ii) is a class of mechanisms whose simplest instance are the pushdown Petri nets and which also naturally subsumes priority multicounter machines (see also Remark 3.7).

Another recent extension of the decidability of reachability of Petri nets has been obtained by Atig and Ganty [2]. In fact, it is a partial solution to the reachability problem for PPN. Their proof also relies on priority multicounter machines. They show that given a *finite-index* context-free language K and a language L generated by a Petri net, it is decidable whether the intersection $K \cap L$ is empty. Note that without the finite-index requirement, this would be equivalent to the reachability problem for PPN. Our final contribution is a decidability result that subsumes both the decidability of model (i) and the result of Atig and Ganty. We present a natural language class that contains both the intersections considered by Atig and Ganty and the languages of model (i) and still has a decidable emptiness problem. To this end, we employ a slightly stronger (and perhaps simpler) version of Atig and Ganty's reduction.

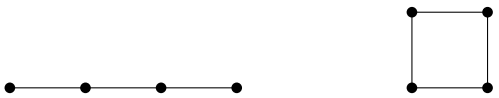
Hence, the perspective of valence automata allows us to identify natural storage mechanisms that (i) push the frontier of decidable emptiness (and hence reachability) and (ii) let us naturally interpret PPN and priority multicounter machines as special cases of a more powerful model that might enjoy decidability, respectively.

The paper is structured as follows. We present the main results in Section 3 and prove them in Sections 4 to 6. Section 4 presents the undecidability part, Section 5 treats the decidable cases, and Section 6 shows the expressive equivalence with the more mechanical descriptions. In Section 7, we present the enhanced decidability result that also subsumes the one by Atig and Ganty.

This work is an extended version of the paper [21]. This version provides proofs of the results of [21] and the enhanced decidability result. Moreover, it contains proofs of some results that first appeared in [19, 20], but have not yet undergone journal peer review.

2. Preliminaries

A *monoid* is a set M together with a binary associative operation such that M contains a neutral element. Unless the monoid at hand warrants a different notation, we will denote the neutral element by 1 and the product of $x, y \in M$ by xy. If X is a set of symbols, X^* denoted the set of words over X. The length of the word $w \in X^*$ is denoted $|w|$. An *alphabet* is a finite set of symbols. The empty word is denoted by $\varepsilon \in X^*$. Let $P \subseteq X \times X$ is a set of pairs of symbols, then the *semi-Dyck language* over P, denoted D_P^* is the smallest subset of X^* such that $\varepsilon \in D_P^*$ and whenever $uv \in \mathcal{D}_P^*$, then also $ua\bar{a}v \in D_P^*$ for every



(a) P4 (b) C4

FIGURE 1. Graphs C4 and P4.

$(a, \bar{a}) \in P$. If $P = \{(a_i, \bar{a}_i) \mid i \in \{1, \dots, n\}\}$, then we also write D_n^* instead of D_P^*. Moreover, if $P = \{(a, b)\}$, then the words in D_P^* are called *semi-Dyck words over* a, b. If $w \in X^*$ is a word with $w = x_1 \cdots x_n$ for $x_1, \dots, x_n \in X$, then w^R denotes w in reverse, i.e. $w^R = x_n \cdots x_1$.

For an alphabet X and languages $L, K \subseteq X^*$, the *shuffle product* $L \sqcup\!\sqcup K$ is the set of all words $u_0 v_1 u_1 \cdots v_n u_n$ where $u_0, \dots, u_n, v_1, \dots, v_n \in X^*$, $u_0 \cdots u_n \in L$, and $v_1 \cdots v_n \in K$. For a subset $Y \subseteq X$, we define the *projection morphism* $\pi_Y \colon X^* \to Y^*$ by $\pi_Y(y) = y$ for $y \in Y$ and $\pi_Y(x) = \varepsilon$ for $x \in X \setminus Y$. Moreover, we define $|w|_Y = |\pi_Y(w)|$ and for $x \in X$, we set $|w|_x = |w|_{\{x\}}$.

Valence automata. As a framework for studying which storage mechanisms permit decidability of the emptiness problem, we employ valence automata. They feature a monoid that dictates which computations are valid. Hence, by an appropriate choice of the monoid, valence automata can be instantiated to be equivalent to a concrete automata model with storage. For the purposes of this work, *equivalent* is meant with respect to accepted languages. Therefore, we regard valence automata as language accepting devices.

Let M be a monoid and X an alphabet. A *valence automaton over* M is a tuple $\mathcal{A} = (Q, X, M, E, q_0, F)$, in which (i) Q is a finite set of *states*, (ii) E is a finite subset of $Q \times X^* \times M \times Q$, called the set of *edges*, (iii) $q_0 \in Q$ is the *initial state*, and (iv) $F \subseteq Q$ is the set of *final states*. For $q, q' \in Q$, $w, w' \in X^*$, and $m, m' \in M$, we write $(q, w, m) \to_{\mathcal{A}} (q', w', m')$ if there is an edge $(q, v, n, q') \in E$ such that $w' = wv$ and $m' = mn$. The language *accepted* by $\mathcal{A}$ is then

$$\mathsf{L}(\mathcal{A}) = \{w \in X^* \mid (q_0, \varepsilon, 1) \to_{\mathcal{A}}^* (f, w, 1) \text{ for some } f \in F\}.$$

The class of languages accepted by valence automata over M is denoted by $\mathsf{VA}(M)$. If $\mathcal{M}$ is a class of monoids, we write $\mathsf{VA}(\mathcal{M})$ for $\bigcup_{M \in \mathcal{M}} \mathsf{VA}(M)$.

Graphs. A *graph* is a pair $\Gamma = (V, E)$ where V is a finite set and E is a subset of $\{S \subseteq V \mid 1 \leq |S| \leq 2\}$. The elements of V are called *vertices* and those of E are called *edges*. Vertices $v, w \in V$ are *adjacent* if $\{v, w\} \in E$. If $\{v\} \in E$ for some $v \in V$, then v is called a *looped* vertex, otherwise it is *unlooped*. A *subgraph* of Γ is a graph (V', E') with $V' \subseteq V$ and $E' \subseteq E$. Such a subgraph is called *induced (by V')* if $E' = \{S \in E \mid S \subseteq V'\}$, i.e. E' contains all edges from E incident to vertices in V'. By $\Gamma \setminus \{v\}$, for $v \in V$, we denote the subgraph of Γ induced by $V \setminus \{v\}$. By C4 (P4), we denote a graph that is a cycle (path) on four vertices; see Fig. 1. Moreover, Γ^- denotes the graph obtained from Γ by deleting all loops: We have $\Gamma^- = (V, E^-)$, where $E^- = \{S \in E \mid |S| = 2\}$. The graph Γ is *loop-free* if $\Gamma^- = \Gamma$. Finally, a *clique* is a loop-free graph in which any two distinct vertices are adjacent.

Products and presentations. If M, N are monoids, then $M \times N$ denotes their *direct product*, whose set of elements is the cartesian product of M and N and composition is defined component-wise. By M^n, we denote the n-fold direct product, i.e. $M \times \cdots \times M$ with n factors.

Let A be a (not necessarily finite) set of symbols and R be a subset of $A^* \times A^*$. The pair (A, R) is called a *(monoid) presentation*. The smallest congruence of the free monoid A^* containing R is denoted by $\equiv_R$ and we will write $[w]_R$ for the congruence class of $w \in A^*$. The *monoid presented by* (A, R) is defined as $A^*/\equiv_R$. Note that since we did not impose a finiteness restriction on A, up to isomorphism, every monoid has a presentation. If

$A = \{a_1, \dots, a_n\}$ and $R = \{(r_i, \bar{r}_i) \mid i \in \{1, \dots, k\}\}$, we also use the shorthand $\langle a_1, \dots, a_n \mid r_1 = \bar{r}_1, \dots, r_k = \bar{r}_k \rangle$ to denote the monoid presented by (A, R).

Furthermore, for monoids M_1, M_2 we can find presentations (A_1, R_1) and (A_2, R_2) such that $A_1 \cap A_2 = \emptyset$. We define the *free product* $M_1 * M_2$ to be presented by $(A_1 \cup A_2, R_1 \cup R_2)$. Note that $M_1 * M_2$ is well-defined up to isomorphism. In analogy to the n-fold direct product, we write $M^{(n)}$ for the n-fold free product of M.

Graph monoids. A presentation (A, R) in which A is a finite alphabet is a *Thue system*. To each graph $\Gamma = (V, E)$, we associate the Thue system $T_\Gamma = (X_\Gamma, R_\Gamma)$ over the alphabet $X_\Gamma = \{a_v, \bar{a}_v \mid v \in V\}$. R_Γ is defined as

$$R_\Gamma = \{(a_v \bar{a}_v, \varepsilon) \mid v \in V\} \cup \{(xy, yx) \mid x \in \{a_v, \bar{a}_v\},\ y \in \{a_w, \bar{a}_w\},\ \{v, w\} \in E\}.$$

In particular, we have $(a_v \bar{a}_v, \bar{a}_v a_v) \in R_\Gamma$ whenever $\{v\} \in E$. To simplify notation, the congruence $\equiv_{R_\Gamma}$ is then also denoted by $\equiv_\Gamma$. We are now ready to define graph monoids. To each graph Γ, we associate the monoid

$$\mathbb{M}\Gamma \ = \ X_\Gamma^* / \equiv_\Gamma.$$

The monoids of the form $\mathbb{M}\Gamma$ are called *graph monoids*.

Storage mechanisms as graph monoids. Let us briefly discuss how to realize storage mechanisms by graph monoids. First, suppose Γ_0 and Γ_1 are disjoint graphs. If Γ is the union of Γ_0 and Γ_1, then $\mathbb{M}\Gamma \cong \mathbb{M}\Gamma_0 * \mathbb{M}\Gamma_1$ by definition. Moreover, if Γ is obtained from Γ_0 and Γ_1 by drawing an edge between each vertex of Γ_0 and each vertex of Γ_1, then $\mathbb{M}\Gamma \cong \mathbb{M}\Gamma_0 \times \mathbb{M}\Gamma_1$.

If Γ consists of one vertex v and has no edges, the only rule in the Thue system is $(a_v \bar{a}_v, \varepsilon)$. In this case, $\mathbb{M}\Gamma$ is also denoted as $\mathbb{B}$ and we will refer to it as the *bicyclic monoid*. The generators a_v and $\bar{a}_v$ are then also written a and $\bar{a}$, respectively. It is not hard to see that $\mathbb{B}$ corresponds to a *partially blind counter*, i.e. one that attains only non-negative values and has to be zero at the end of the computation. Moreover, if Γ consists of one looped vertex, then $\mathbb{M}\Gamma$ is isomorphic to $\mathbb{Z}$ and thus realizes a *blind counter*, which can go below zero and is zero-tested in the end.

If one storage mechanism is realized by a monoid M, then the monoid $\mathbb{B} * M$ corresponds to the mechanism that *builds stacks*: A configuration of this new mechanism consists of a sequence $c_0 a c_1 \cdots a c_n$, where $c_0, \dots, c_n$ are configurations of the mechanism realized by M. We interpret this as a stack with the entries $c_0, \dots, c_n$. One can open a new stack entry on top (by multiplying $a \in \mathbb{B}$), remove the topmost entry if empty (by multiplying $\bar{a} \in \mathbb{B}$) and operate on the topmost entry using the old mechanism (by multiplying elements from M). In particular, $\mathbb{B} * \mathbb{B}$ describes a pushdown stack with two stack symbols. See [22] for more examples and [23] for more details.

As a final example, suppose Γ is one edge short of being a clique, then $\mathbb{M}\Gamma \cong \mathbb{B}^{(2)} \times \mathbb{B}^{n-2}$, where n is the number of vertices in Γ. Then, by the observations above, valence automata over $\mathbb{M}\Gamma$ are equivalent to Petri nets with $n-2$ unbounded places and access to a pushdown stack. Hence, for our purposes, a *pushdown Petri net* is a valence automaton over $\mathbb{B}^{(2)} \times \mathbb{B}^n$ for some $n \in \mathbb{N}$.

3. Results

As a first step, we exhibit graphs Γ for which $\mathsf{VA}(\mathbb{M}\Gamma)$ includes the recursively enumerable languages.

Theorem 3.1. *Let Γ be a graph such that Γ^- contains $\mathsf{C4}$ or $\mathsf{P4}$ as an induced subgraph. Then $\mathsf{VA}(\mathbb{M}\Gamma)$ is the class of recursively enumerable languages. In particular, the emptiness problem is undecidable for valence automata over $\mathbb{M}\Gamma$.*

This unifies and slightly strengthens a few undecidability results concerning valence automata over graph monoids. The case that all vertices are looped was shown by Lohrey and Steinberg [14] (see also the discussion of Theorem 3.4). Another case appeared in [19]. We prove Theorem 3.1 in Section 4.

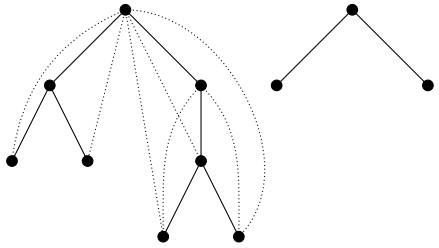
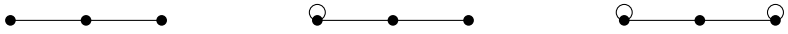


FIGURE 2. Example of a transitive forest. The solid edges are part of the trees whose comparability graphs make up the graph. The transitive forest consists of both the solid and the dashed edges.

It is not clear whether Theorem 3.1 describes all Γ for which $\mathsf{VA}(\mathbb{M}\Gamma)$ exhausts the recursively enumerable languages. For example, as mentioned above, if Γ is one edge short of being a clique, then valence automata over $\mathbb{M}\Gamma$ are pushdown Petri nets. In particular, the emptiness problem for valence automata is equivalent to the reachability problem of this model, for which decidability is a long-standing open question [15]. In fact, it is already open whether reachability is decidable in the case of $\mathbb{B}^{(2)} \times \mathbb{B}$, although Leroux, Sutre, and Totzke have recently made progress on this case [12]. Therefore, characterizing those Γ with a decidable emptiness problem for valence automata over $\mathbb{M}\Gamma$ would very likely settle these open questions[2].

However, we will show that if we steer clear of pushdown Petri nets, we can achieve a characterization. More precisely, we will present a set of graphs that entail the behavior of pushdown Petri nets. Then, we show that among those graphs that do not contain these as induced subgraphs, the absence of $\mathsf{P4}$ and $\mathsf{C4}$ already characterizes decidability.

PPN-graphs. A graph Γ is said to be a *PPN-graph* if it is isomorphic to one of the following three graphs:



We say that the graph Γ is *PPN-free* if it has no PPN-graph as an induced subgraph. Observe that a graph Γ is PPN-free if and only if in the neighborhood of each unlooped vertex, any two vertices are adjacent.

Of course, the abbreviation 'PPN' refers to 'pushdown Petri nets'. This is justified by the following fact. It is proven in Section 5 (page 13).

Proposition 3.2. *If Γ is a PPN-graph, then $\mathsf{VA}(\mathbb{M}\Gamma) = \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{B})$.*

Transitive forests. In order to exploit the absence of $\mathsf{P4}$ and $\mathsf{C4}$ as induced subgraphs, we will employ a characterization of such graphs as transitive forests. The *comparability graph* of a tree t is a simple graph with the same vertices as t, but has an edge between two vertices whenever one is a descendant of the other in t. A graph Γ is a *transitive forest* if the simple graph Γ^- is a disjoint union of comparability graphs of trees. For an example of a transitive forest, see Fig. 2.

Let DEC denote the smallest isomorphism-closed class of monoids such that

1. for each $n \geq 0$, we have $\mathbb{B}^n \in \mathsf{DEC}$ and
2. for $M, N \in \mathsf{DEC}$, we also have $M * N \in \mathsf{DEC}$ and $M \times \mathbb{Z} \in \mathsf{DEC}$.

Our main result characterizes those PPN-free Γ for which valence automata over $\mathbb{M}\Gamma$ have a decidable emptiness problem.

Theorem 3.3. *Let Γ be PPN-free. Then the following conditions are equivalent:*

[2]Strictly speaking, it is conceivable that there is a decision procedure for each $\mathbb{B}^{(2)} \times \mathbb{B}^n$, but no uniform one that works for all n. However, this seems unlikely.

1. *Emptiness is decidable for valence automata over* $\mathbb{M}\Gamma$.
2. Γ^- *contains neither* $\mathsf{C4}$ *nor* $\mathsf{P4}$ *as an induced subgraph.*
3. Γ *is a transitive forest.*
4. $\mathbb{M}\Gamma \in \mathsf{DEC}$.

We present the proof in Section 5. Note that this generalizes the fact that emptiness is decidable for pushdown automata (i.e. graphs with no edges) and partially blind multicounter automata (i.e. cliques), or equivalently, reachability in Petri nets.

Note that if Γ has a loop on every vertex, then $\mathbb{M}\Gamma$ is a group. Groups that arise in this way are called *graph groups*. In general, if a monoid M is a group, then emptiness for valence automata over M is decidable if and only if the *rational subset membership problem* is decidable for M [11]. The latter problem asks, given a rational set R over M and an element $m \in M$, whether $m \in R$; see [13] for more information. Therefore, Theorem 3.3 extends the following result of Lohrey and Steinberg [14], which characterizes those graph groups for which the rational subset membership problem is decidable.

Theorem 3.4 (Lohrey and Steinberg [14])**.** *Let* Γ *be a graph in which every vertex is looped. Then the rational subset membership problem for the group* $\mathbb{M}\Gamma$ *is decidable if and only if* Γ *is a transitive forest.*

Lohrey and Steinberg show decidability by essentially proving that in their case, the languages in $\mathsf{VA}(\mathbb{M}\Gamma)$ have semilinear Parikh images (although they use different terminology). Here, we extend this argument by showing that in the equivalent cases of Theorem 3.3, the Parikh images of $\mathsf{VA}(\mathbb{M}\Gamma)$ are those of languages accepted by priority multicounter machines. The latter were introduced and shown to have a decidable reachability problem by Reinhardt [15].

Intuition for decidable cases. In order to provide an intuition for those storage mechanisms (not containing a pushdown Petri net) with a decidable emptiness problem, we present an equally expressive class of monoids for which the corresponding storage mechanisms are easier to grasp. Let $\mathsf{SC}^{\pm}$ be the smallest isomorphism-closed class of monoids with

1. for each $n \in \mathbb{N}$, we have $\mathbb{B}^n \in \mathsf{SC}^{\pm}$,
2. for each $M \in \mathsf{SC}^{\pm}$, we also have $\mathbb{B} * M \in \mathsf{SC}^{\pm}$ and $M \times \mathbb{Z} \in \mathsf{SC}^{\pm}$.

Thus, $\mathsf{SC}^{\pm}$ realizes those storage mechanisms that can be constructed from a finite set of *partially blind counters* ($\mathbb{B}^n$) by *building stacks* ($M \mapsto \mathbb{B} * M$) and *adding blind counters* ($M \mapsto M \times \mathbb{Z}$). Then, in fact, the monoids in $\mathsf{SC}^{\pm}$ produce the same languages as those in DEC.

Proposition 3.5. $\mathsf{VA}(\mathsf{DEC}) = \mathsf{VA}(\mathsf{SC}^{\pm})$.

Proposition 3.5 is proven in Section 6. While our decidability proof for $\mathsf{SC}^{\pm}$ will be a reduction to priority multicounter machines (see Section 5 for a definition), it seems likely that these two models are incomparable in terms of expressiveness (see the remarks after Theorem 5.12).

Intersections with finite-index languages. This work exhibits valence automata over $\mathsf{SC}^{\pm}$ as an extension of Petri nets that features a type of stack but retains decidability of the emptiness problem. Another recent result of this kind has been obtained by Atig and Ganty [2]. They showed that given a finite-index context-free language K and a Petri net language L, it is decidable whether $K \cap L$ is empty. Moreover, they also employ a reduction to priority multicounter machines. This raises the question of how the two results relate to each other. In Section 7, we present a natural language class that subsumes both the languages of Atig and Ganty and those of $\mathsf{VA}(\mathsf{SC}^{\pm})$ and prove that emptiness is still decidable. Intuitively, this class is obtained by taking languages of Atig and Ganty and then applying operators corresponding to *building stacks* and *adding blind counters*. The precise definition and the result can be found in Section 7.

Intuition for open cases. We also want to provide an intuition for the remaining storage mechanisms, i.e. those defined by monoids $\mathbb{M}\Gamma$ about which Theorems 3.1 and 3.3 make no statement. To this end, we describe a class of monoids that are expressively equivalent to these remaining cases. The remaining cases are given by those graphs Γ where Γ^- does not contain $\mathsf{C4}$ or $\mathsf{P4}$, but Γ contains a PPN-graph. Let REM denote the class of monoids $\mathbb{M}\Gamma$, where Γ is such a graph. Let SC^+ be the smallest isomorphism-closed class of monoids with

1. $\mathbb{B}^{(2)} \times \mathbb{B} \in \mathsf{SC}^+$ and
2. for each $M \in \mathsf{SC}^+$, we also have $\mathbb{B} * M \in \mathsf{SC}^+$ and $M \times \mathbb{B} \in \mathsf{SC}^+$.

This means, SC^+ realizes those storage mechanisms that are obtained from *a pushdown stack, together with one partially blind counter* ($\mathbb{B}^{(2)} \times \mathbb{B}$) by the transformations of *building stacks* ($M \mapsto \mathbb{B} * M$) and *adding partially blind counters* ($M \mapsto M \times \mathbb{B}$).

Proposition 3.6. $\mathsf{VA}(\mathsf{REM}) = \mathsf{VA}(\mathsf{SC}^+)$.

We prove Proposition 3.6 in Section 6. Of course, SC^+ generalizes pushdown Petri nets, which correspond to monoids $\mathbb{B}^{(2)} \times \mathbb{B}^n$ for $n \in \mathbb{N}$. Moreover, SC^+ also subsumes priority multicounter machines (see p. 14 for a definition) in a straightforward way: Every time we build stacks, we can use the new pop operation to realize a zero test on all the counters we have added so far. Let $M_0 = \mathbf{1}$ and $M_{k+1} = \mathbb{B} * (M_k \times \mathbb{B})$. Then, priority k-counter machines correspond to valence automata over M_k where the stack heights never exceed 1.

Remark 3.7. Priority multicounter machines are already subsumed by pushdown Petri nets alone: Atig and Ganty [2, Lemma 7] show implicitly that for each priority multicounter machine, one can construct a pushdown Petri net that accepts the same language. Hence, valence automata over SC^+ are not the first perhaps-decidable generalization of both pushdown Petri nets and priority multicounter machines, but they generalize both in a natural way.

4. Undecidability

In this section, we prove Theorem 3.1. It should be mentioned that a result similar to Theorem 3.1 was shown by Lohrey and Steinberg [14]: They proved that if every vertex in Γ is looped and Γ^- contains $\mathsf{C4}$ or $\mathsf{P4}$ as an induced subgraph, then the rational subset membership problem is undecidable for $\mathbb{M}\Gamma$. Their proof adapts a construction of Aalbersberg and Hoogeboom [1], which shows that the disjointness problem for rational sets of traces is undecidable when the independence relation has $\mathsf{P4}$ or $\mathsf{C4}$ as an induced subgraph. An inspection of the proof presented here, together with its prerequisites (Theorems 4.2 and 4.3), reveals that the employed ideas are very similar to the combination of Lohrey and Steinberg's and Aalbersberg and Hoogeboom's proof.

A *language class* is a collection of languages that contains at least one non-empty language. In this work, for each language class, there is a way to finitely represent each member of the class. Moreover, an inclusion $\mathcal{C} \subseteq \mathcal{D}$ between language classes $\mathcal{C}$ and $\mathcal{D}$ is always meant to be *effective*, in other words: Given a representation of a language in $\mathcal{C}$, we can compute a representation of that language in $\mathcal{D}$. The same holds for equalities between language classes.

Let X and Y be alphabets. A relation $T \subseteq X^* \times Y^*$ is called a *rational transduction* if there is an alphabet W, a regular language $R \subseteq W^*$, and morphisms $g\colon W^* \to X^*$ and $h\colon W^* \to Y^*$ such that $T = \{(g(w), h(w)) \mid w \in R\}$ (see [3]). For a language $L \subseteq X^*$, we define $TL = \{v \in Y^* \mid \exists u \in L\colon (u, v) \in T\}$. A language class $\mathcal{C}$ is a *full trio* if for every language L in $\mathcal{C}$, the language TL is effectively contained in $\mathcal{C}$ as well. Here, "effectively" means again that given a representation of a language L from $\mathcal{C}$ and a description of T, one can effectively compute a representation of TL. For a language L, we denote by $\mathcal{T}(L)$ the smallest full trio containing L. Note that if $L \neq \emptyset$, the class $\mathcal{T}(L)$ contains precisely the languages TL for rational transductions T. For example, it is well-known that for every monoid M, the class $\mathsf{VA}(M)$ is a full trio [6]. A *full AFL* is a full trio that is also closed

under Kleene iteration, i.e. for each member L, the language L^* is effectively a member as well.

Here, we use the following fact. We denote the recursively enumerable languages by RE.

Lemma 4.1. *Let $X = \{a_1, \bar{a}_1, b_1, a_2, \bar{a}_2, b_2\}$ and let $B_2 \subseteq X^*$ be defined as*

$$B_2 = (\{a_1^n \bar{a}_1^n \mid n \geq 0\}b_1)^* \sqcup (\{a_2^n \bar{a}_2^n \mid n \geq 0\}b_2)^*.$$

Then RE equals $\mathcal{T}(B_2)$, the smallest full trio containing B_2.

Lemma 4.1 is essentially due to Hartmanis and Hopcroft, who stated it in slightly different terms:

Theorem 4.2 (Hartmanis and Hopcroft [9])**.** *Let $\mathcal{C}$ be the smallest full AFL containing $\{a^n b^n \mid n \geq 0\}$. Every recursively enumerable language is the homomorphic image of the intersection of two languages in $\mathcal{C}$.*

By the following auxiliary result of Ginsburg and Greibach [8, Theorem 3.2a], Lemma 4.1 will follow from Theorem 4.2.

Theorem 4.3 (Ginsburg and Greibach [8])**.** *Let $L \subseteq X^*$ and $c \notin X$. The smallest full AFL containing L equals $\mathcal{T}((Lc)^*)$.*

As announced, Lemma 4.1 now follows.

Lemma 4.1. Since clearly $\mathcal{T}(B_2) \subseteq \mathsf{RE}$, it suffices to show $\mathsf{RE} \subseteq \mathcal{T}(B_2)$. According to Theorem 4.2, this amounts to showing that $L_1 \cap L_2 \in \mathcal{T}(B_2)$ for any L_1 and L_2 in $\mathcal{C}$, where $\mathcal{C}$ is the smallest full AFL containing the language $S = \{a^n b^n \mid n \geq 0\}$. Hence, let $L_1, L_2 \in \mathcal{C}$. By Theorem 4.3, L_1 and L_2 belong to $\mathcal{C} = \mathcal{T}((Sc)^*)$. This means we have $L_i = T_i(\{a_i^n \bar{a}_i^n \mid n \geq 0\}b_i)^*$ for some rational transduction T_i for $i = 1, 2$. Using a product construction, it is now easy to obtain a rational transduction T with $TB_2 = L_1 \cap L_2$. □

The proof of Theorem 3.1 will require one more auxiliary lemma. In the following, $[w]_\Gamma$ denotes the congruence class of $w \in X_\Gamma^*$ with respect to $\equiv_\Gamma$.

Lemma 4.4. *Let $\Gamma = (V, E)$ be a graph, let $W \subseteq V$ be a subset of vertices, and let $Y \subseteq X_\Gamma$ be defined as $Y = \{a_w, \bar{a}_w \mid w \in W\}$. Then $u \equiv_\Gamma v$ implies $\pi_Y(u) \equiv_\Gamma \pi_Y(v)$ for $u, v \in X_\Gamma^*$.*

Proof. An inspection of the rules in the Thue system T_Γ reveals that if $(u, v) \in R_\Gamma$, then either $(\pi_Y(u), \pi_Y(v)) = (u, v)$ or $\pi_Y(u) = \pi_Y(v)$. In any case, $\pi_Y(u) \equiv_\Gamma \pi_Y(v)$. Since $\equiv_\Gamma$ is a congruence and π_Y a morphism, this implies the lemma. □

Note that the foregoing lemma does not hold for arbitrary alphabets $Y \subseteq X_\Gamma$. For example, if $V = \{1\}$, $X_\Gamma = \{a_1, \bar{a}_1\}$, and $Y = \{a_1\}$, then $a_1\bar{a}_1 \equiv_\Gamma \varepsilon$, but $a_1 \not\equiv_\Gamma \varepsilon$.

We are now ready to prove Theorem 3.1.

Theorem 3.1. Observe that $w \equiv_\Gamma \varepsilon$ if and only if w can be transformed into ε by finitely many times replacing an infix u with an infix v for some $(u, v) \in R_\Gamma$. Since R_Γ is finite, this implies that the set of all $w \in X_\Gamma^*$ with $w \equiv_\Gamma \varepsilon$ is recursively enumerable. (In fact, whether $w \equiv_\Gamma \varepsilon$ can be decided in polynomial time [19, 23].) In particular, one can recursively enumerate runs of valence automata over $\mathsf{VA}(\mathbb{M}\Gamma)$ and hence $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{RE}$. For the other inclusion, recall that $\mathsf{VA}(M)$ is a full trio for any monoid M. Furthermore, if Δ is an induced subgraph of Γ, then $\mathbb{M}\Delta$ embeds into $\mathbb{M}\Gamma$, meaning $\mathsf{VA}(\mathbb{M}\Delta) \subseteq \mathsf{VA}(\mathbb{M}\Gamma)$. Hence, according to Lemma 4.1, it suffices to show that $B_2 \in \mathsf{VA}(\mathbb{M}\Gamma)$ if Γ^- equals $\mathsf{C4}$ or $\mathsf{P4}$.

Let $X = \{a_1, \bar{a}_1, b_1, a_2, \bar{a}_2, b_2\}$. and $\Gamma = (V, E)$. If Γ^- equals $\mathsf{C4}$ or $\mathsf{P4}$, then $V = \{1, 2, 3, 4\}$ with $\{3, 1\}, \{1, 2\}, \{2, 4\} \in E$ and $\{1, 4\}, \{2, 3\} \notin E$. See Fig. 3. We construct a valence automaton $\mathcal{A}$ over $\mathbb{M}\Gamma$ for $B_2 \subseteq X^*$ as follows. First, $\mathcal{A}$ reads a word in $R = ((a_1^*\bar{a}_1^*)b_1)^* \sqcup ((a_2^*\bar{a}_2^*)b_2)^*$. Here, when reading a_i or $\bar{a}_i$, it multiplies $[a_i]$ or $[\bar{a}_i]$, respectively, to the storage monoid. When reading b_1 or b_2, it multiplies $[a_4]$ or $[a_3]$, respectively. After this, $\mathcal{A}$ switches to another state and nondeterministically multiplies an

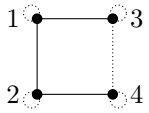


FIGURE 3. Graphs Γ where Γ^- is C4 or P4. Dotted lines represent edges that may or may not exist in Γ.

element from $\{[\bar{a}_4], [\bar{a}_3]\}^*$. Then it changes into an accepting state. We shall prove that $\mathcal{A}$ accepts B_2. Let the morphism $h\colon X^* \to \{a_i, \bar{a}_i \mid 1 \leq i \leq 4\}^*$ be defined by $h(a_i) = a_i$ and $h(\bar{a}_i) = \bar{a}_i$ for $i = 1, 2$ and $h(b_1) = a_4$ and $h(b_2) = a_3$.

Suppose $w \in \mathsf{L}(\mathcal{A})$. Then $w \in R$ and there is a $v \in \{\bar{a}_4, \bar{a}_3\}^*$ with $[h(w)v]_\Gamma = [\varepsilon]_\Gamma$. Let $w_i = \pi_{\{a_i, \bar{a}_i, b_i\}}(w)$. If we can show $w_i \in (\{a_i^n \bar{a}_i^n \mid n \geq 0\}^* b_i)^*$ for $i = 1, 2$, then clearly $w \in B_2$. For symmetry reasons, it suffices to prove this for $i = 1$. Let $Y = \{a_1, \bar{a}_1, a_4, \bar{a}_4\}$. Since $[h(w)v]_\Gamma = [\varepsilon]_\Gamma$, we have in particular $[\pi_Y(h(w)v)]_\Gamma = [\varepsilon]_\Gamma$ by Lemma 4.4. Moreover,

$$\pi_Y(h(w)v) = a_1^{n_1} \bar{a}_1^{\bar{n}_1} a_4 \cdots a_1^{n_k} \bar{a}_1^{\bar{n}_k} a_4 \bar{a}_4^m$$

for some $n_1, \ldots, n_k, \bar{n}_1, \ldots, \bar{n}_k, m \in \mathbb{N}$. Again, by projecting to $\{a_4, \bar{a}_4\}^*$, we obtain $[a_4^k \bar{a}_4^m]_\Gamma = [\varepsilon]_\Gamma$ and hence $k = m$. If $n_k \neq \bar{n}_k$, then it is easy to see that $\pi_Y(h(w)v)$ cannot be reduced to ε, since there is no edge $\{1, 4\}$ in Γ. Therefore, we have $n_k = \bar{n}_k$. It follows inductively that $n_i = \bar{n}_i$ for all $1 \leq i \leq k$. Since $w_i = a_1^{n_1} \bar{a}_1^{\bar{n}_1} b_1 \cdots a_1^{n_k} \bar{a}_1^{\bar{n}_k} b_1$, this implies $w_i \in (\{a_1^n \bar{a}_1^n \mid n \geq 0\} b_1)^*$.

We shall now prove $B_2 \subseteq \mathsf{L}(\mathcal{A})$. Let $g\colon X^* \to \{\bar{a}_3, \bar{a}_4\}$ be the morphism defined by $g(a_i) = g(\bar{a}_i) = \varepsilon$ and $g(b_1) = \bar{a}_4$ and $g(b_2) = \bar{a}_3$. We show by induction on $|w|$ that $w \in B_2$ implies $[h(w)g(w)^R]_\Gamma = [\varepsilon]_\Gamma$. Since for each $w \in B_2$, $\mathcal{A}$ clearly has a run that puts $[h(w)g(w)^R]_\Gamma$ into the storage, this establishes $B_2 \subseteq \mathsf{L}(\mathcal{A})$. Suppose $\pi_{\{b_1, b_2\}}(w)$ ends in b_1. Then $w = rsb_1$ for some $r \in X^*$, $s \in (a_1^n \bar{a}_1^n) \sqcup t$ with $n \in \mathbb{N}$ and $t \in \{a_2, \bar{a}_2, b_2\}^*$. Note that then $rt \in B_2$. Since there are edges $\{1, 2\}, \{1, 3\}$ in Γ, we have $[h(s)]_\Gamma = [h(ta_1^n \bar{a}_1^n)]_\Gamma$. Moreover, since g deletes a_1 and $\bar{a}_1$, we have $g(s) = g(t)$. Therefore,

$$\begin{aligned}[h(w)g(w)^R]_\Gamma &= [h(rsb_1)g(rsb_1)^R]_\Gamma = [h(rta_1^n \bar{a}_1^n b_1)g(rtb_1)^R]_\Gamma \\ &= [h(rt)a_1^n \bar{a}_1^n a_4 \bar{a}_4 g(rt)^R]_\Gamma = [h(rt)g(rt)^R]_\Gamma.\end{aligned}$$

By induction, we have $[h(rt)g(rt)^R]_\Gamma = [\varepsilon]_\Gamma$ and hence $[h(w)g(w)^R]_\Gamma = [\varepsilon]_\Gamma$. If $\pi_{\{b_1, b_2\}}(w)$ ends in b_2, then one can show $[h(w)g(w)^R]_\Gamma = [\varepsilon]_\Gamma$ completely analogously. This proves $B_2 \subseteq \mathsf{L}(\mathcal{A})$ and hence the theorem. □

5. Decidability

In this section, we prove Theorem 3.3 and Proposition 3.2. First, we mention existing results that are ingredients to our proofs.

Let $\mathcal{C}$ be a class of languages. A *$\mathcal{C}$-grammar* is a quadruple $G = (N, T, P, S)$ where N and T are disjoint alphabets and $S \in N$. P is a finite set of pairs (A, M) with $A \in N$ and $M \subseteq (N \cup T)^*$, $M \in \mathcal{C}$. A pair $(A, M) \in P$ is called a *production of G*. We write $x \Rightarrow_G y$ if $x = uAv$ and $y = uwv$ for some $u, v, w \in (N \cup T)^*$ and $(A, M) \in P$ with $w \in M$. Moreover, $x \Rightarrow_G^n y$ means that there are $x_0, \ldots, x_n \in (N \cup T)^*$ with $x_{i-1} \Rightarrow_G x_i$ for $1 \leq i \leq n$ and $x_0 = x$ and $x_n = y$. Furthermore, we have $x \Rightarrow_G^* y$ if $x \Rightarrow_G^n y$ for some $n \geq 0$. The *language generated by G* is $\mathsf{L}(G) = \{w \in T^* \mid S \Rightarrow_G^* w\}$. The class of all languages that are generated by $\mathcal{C}$-grammars is called the *algebraic extension of $\mathcal{C}$* and is denoted $\mathsf{Alg}(\mathcal{C})$. Of course, if $\mathcal{C} \subseteq \mathcal{D}$, then $\mathsf{Alg}(\mathcal{C}) \subseteq \mathsf{Alg}(\mathcal{D})$. Moreover, it is easy to see that if $\mathcal{C} \subseteq \mathsf{Alg}(\mathcal{D})$, then $\mathsf{Alg}(\mathcal{C} \cup \mathcal{D}) = \mathsf{Alg}(\mathcal{D})$.

The following is easy to show in the same way one shows that the context-free languages constitute a full trio [3]. A proof can be found in [23].

Lemma 5.1. *If $\mathcal{C}$ is a full trio, then $\mathsf{Alg}(\mathcal{C})$ is a full trio as well.*

A monoid M is called *finitely generated* if there is a finite subset $F \subseteq M$ such that every element of M can be written as a product of elements of F. A language $I \subseteq X^*$ is called an *identity language for M* if there is a surjective morphism $\varphi\colon X^* \to M$ with $I = \varphi^{-1}(1)$. We will also use the following well-known fact about valence automata. A proof can be found, e.g., in [23, 10].

Proposition 5.2. *Let M be a finitely generated monoid. Then:*

(1) $\mathsf{VA}(M)$ is the smallest full trio containing all identity languages of M.
(2) If L is any identity language of M, then $\mathsf{VA}(M)$ is the smallest full trio containing L.

The well-known theorem of Chomsky and Schützenberger [3], expressed in terms of valence automata, states that $\mathsf{VA}(\mathbb{Z} * \mathbb{Z})$ is the class of context-free languages. This formulation, along with a new proof, is due to Kambites [10]. Let Reg and CF denote the class of regular and context-free languages, respectively. Then we have $\mathsf{Reg} = \mathsf{VA}(\mathbf{1})$ and $\mathsf{CF} = \mathsf{Alg}(\mathsf{Reg})$. Here, $\mathbf{1}$ denotes the trivial monoid $\{1\}$. Moreover, notice that $\mathsf{Alg}(\mathsf{Alg}(\mathcal{C})) = \mathsf{Alg}(\mathcal{C})$ for every language class $\mathcal{C}$. Since furthermore valence automata over $\mathbb{B} * \mathbb{B}$ are equivalent to pushdown automata, we have in summary:

$$\mathsf{CF} = \mathsf{VA}(\mathbb{B} * \mathbb{B}) = \mathsf{Alg}(\mathsf{VA}(\mathbf{1})) = \mathsf{Alg}(\mathsf{CF}) = \mathsf{VA}(\mathbb{Z} * \mathbb{Z}). \tag{1}$$

In order to work with general free products, we use the following result, which expresses the languages in $\mathsf{VA}(M_0 * M_1)$ in terms of $\mathsf{VA}(M_0)$ and $\mathsf{VA}(M_1)$. It was first shown in [19]. In [4], it was extended to more general products. For the convenience of the reader, we include a proof.

Proposition 5.3 ([19])**.** *Let M_0 and M_1 be monoids. Then $\mathsf{VA}(M_0 * M_1)$ is included in $\mathsf{Alg}(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1))$.*

Proof. For every monoid M, we have $\mathsf{VA}(M) = \bigcup_N \mathsf{VA}(N)$, where N ranges over the finitely generated submonoids of M. Moreover, every finitely generated submonoid of $M_0 * M_1$ is included in some $N_0 * N_1$, where N_i is a finitely generated submonoid of M_i, for $i = 0, 1$. Therefore, we have $\mathsf{VA}(M_0 * M_1) = \bigcup_{N_0, N_1} \mathsf{VA}(N_0 * N_1)$, where N_i ranges over the finitely generated submonoids of M_i, for $i = 0, 1$. Thus, it suffices to show the proposition in the case that M_0 and M_1 are finitely generated.

For $i = 0, 1$, let (A_i, R_i) be a presentation of M_i such that A_i is finite. Then $M_0 * M_1$ is presented by $(A_0 \cup A_1, R_0 \cup R_1)$. Consider the languages $L_i = \{w \in A_i^* \mid w \equiv_{R_i} \varepsilon\}$ for $i \in \{0, 1\}$. Then L_i is an identity language of M_i and hence contained in $\mathsf{VA}(M_i)$. Moreover, by definition of $M_0 * M_1$, the language $L = \{w \in (A_0 \cup A_1)^* \mid w \equiv_{R_0 \cup R_1} \varepsilon\}$ is an identity language of $M_0 * M_1$.

According to Lemma 5.1, the class $\mathsf{Alg}(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1))$ is a full trio. Thus, Proposition 5.2 tells us that it suffices to show that the identity language L of $M_0 * M_1$ is contained in $\mathsf{Alg}(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1))$.

Consider the binary relation $\rightharpoonup$ on $(A_0 \cup A_1)^*$ where $u \rightharpoonup v$ if and only if for some $i \in \{0, 1\}$, there are $x, z \in (A_0 \cup A_1)^*$, $y \in A_i^*$, such that $u = xz$, $v = xyz$, and $y \equiv_{R_i} \varepsilon$. It is now easy to see that $w \equiv_{R_0 \cup R_1} \varepsilon$ if and only if $\varepsilon \rightharpoonup^* w$.

This allows us to construct a $\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1)$-grammar for L. Let $G = (N, A_0 \cup A_1, P, S)$, where $N = \{S\}$. In order to describe the productions, we need to define two languages. For $i \in \{0, 1\}$, let

$$K_i = \{Sa_1S \cdots a_nS \mid a_1, \ldots, a_n \in A_i,\ a_1 \cdots a_n \in L_i\}.$$

Then K_i can be obtained from L_i using full trio operations and is thus contained in $\mathsf{VA}(M_i)$. Our grammar contains only three productions: $S \to K_0$, $S \to K_1$, and $S \to \{\varepsilon\}$ (recall that as a regular language, $\{\varepsilon\}$ belongs to each $\mathsf{VA}(M_i)$). Then, it is immediate that $w \in \mathsf{L}(G)$ if and only if $\varepsilon \rightharpoonup^* w$ and hence $\mathsf{L}(G) = L$. □

Proposition 5.3 tells us that the languages in $\mathsf{VA}(M_0 * M_1)$ are confined to the algebraic extension of $\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1)$. Our next ingredient, Proposition 5.6, will complement Proposition 5.3 by describing monoids N such that the algebraic extension of $\mathsf{VA}(M)$ is confined to $\mathsf{VA}(N)$. We need two auxiliary lemmas, for which the following notation will be convenient. We write $M \hookrightarrow N$ for monoids M, N if there is a morphism $\varphi\colon M \to N$ such that $\varphi^{-1}(1) = \{1\}$. Clearly, if $M \hookrightarrow N$, then $\mathsf{VA}(M) \subseteq \mathsf{VA}(N)$: Replacing in a valence automaton over M all elements $m \in M$ with $\varphi(m)$ yields a valence automaton over N that accepts the same language.

Lemma 5.4. *If $M \hookrightarrow M'$ and $N \hookrightarrow N'$, then we have $M * N \hookrightarrow M' * N'$.*

Proof. Let $\varphi\colon M \to M'$ and $\psi\colon N \to N'$ be morphisms with $\varphi^{-1}(1) = \{1\}$ and $\psi^{-1}(1) = \{1\}$. Then defining $\kappa\colon M * N \to M' * N'$ as the morphism with $\kappa|_M = \varphi$ and $\kappa|_N = \psi$ clearly yields $\kappa^{-1}(1) = 1$. □

For a monoid M, we define $R_1(M) = \{x \in M \mid \exists y \in M\colon xy = 1\}$. Observe that the set $R_1(M)$ can be thought of as the storage contents that can occur in a valid run of a valence automaton over M. The following result appeared first in [20]. We include a proof for the convenience of the reader.

Lemma 5.5 ([20])**.** *Let M be a monoid with $R_1(M) \neq \{1\}$. Then we have $\mathbb{B}^{(n)} * M \hookrightarrow \mathbb{B} * M$ for every $n \geq 1$. In particular, $\mathsf{VA}(\mathbb{B} * M) = \mathsf{VA}(\mathbb{B}^{(n)} * M)$ for every $n \geq 1$.*

Proof. Observe that if $\mathbb{B}^{(n)} * M \hookrightarrow \mathbb{B} * M$ and $\mathbb{B} * \mathbb{B} * M \hookrightarrow \mathbb{B} * M$, then

$$\mathbb{B}^{(n+1)} * M \cong \mathbb{B} * (\mathbb{B}^{(n)} * M) \hookrightarrow \mathbb{B} * (\mathbb{B} * M) \hookrightarrow \mathbb{B} * M.$$

Therefore, it suffices to prove $\mathbb{B} * \mathbb{B} * M \hookrightarrow \mathbb{B} * M$.

Let $\mathbb{B}_s = \langle s, \bar{s} \mid s\bar{s} = 1\rangle$ for $s \in \{p, q, r\}$. We show $\mathbb{B}_p * \mathbb{B}_q * M \hookrightarrow \mathbb{B}_r * M$. Suppose M is presented by (X, R). We regard the monoids $\mathbb{B}_p * \mathbb{B}_q * M$ and $\mathbb{B}_r * M$ as embedded into $\mathbb{B}_p * \mathbb{B}_q * \mathbb{B}_r * M$, which by definition of the free product, has a presentation (Y, S), where $Y = \{p, \bar{p}, q, \bar{q}, r, \bar{r}\} \cup X$ and S consists of R and the equations $s\bar{s} = 1$ for $s \in \{p, q, r\}$. For $w \in Y^*$, we write $[w]$ for the class of w in the congruence generated by S. Since $R_1(M) \neq \{1\}$, we find $u, v \in X^*$ with $[uv] = 1$ and $[u] \neq 1$.

Observe that then for any $f, g \in (\{r, \bar{r}\} \cup X)^*$, we have $[frv\bar{r}g] \neq 1$: By induction on the number of rewriting steps, one can show that every word in $[frv\bar{r}g]$ is of the form $f'rv'\bar{r}g'$ for $f', g' \in (\{r, \bar{r}\} \cup X)^*$ and $v' \in X^*$ with $v' \equiv_R v$. By the same argument, we have $[fru\bar{r}g] \neq 1$ for any $f, g \in (\{r, \bar{r}\} \cup X)^*$.

Let $\varphi\colon (\{p, \bar{p}, q, \bar{q}\} \cup X)^* \to (\{r, \bar{r}\} \cup X)^*$ be the morphism with $\varphi(x) = x$ for $x \in X$ and

$$\begin{aligned} p &\mapsto rr, & \bar{p} &\mapsto \bar{r}\bar{r},\\ q &\mapsto rur, & \bar{q} &\mapsto \bar{r}v\bar{r}. \end{aligned}$$

We show by induction on $|w|$ that $[\varphi(w)] = 1$ implies $[w] = 1$. Since this is trivial for $w = \varepsilon$, we assume $|w| \geq 1$. Now suppose $[\varphi(w)] = [\varepsilon]$ for some $w \in (\{p, \bar{p}, q, \bar{q}\} \cup X)^*$. If $w \in X^*$, then $[\varphi(w)] = [w]$ and hence $[w] = 1$. Otherwise, we have $\varphi(w) = xry\bar{r}z$ for some $y \in X^*$ with $[y] = 1$ and $[xz] = 1$. This means $w = fs\overline{y}\overline{s'}g$ for $s, s' \in \{p, q\}$ with $\varphi(fs) = xr$ and $\varphi(\overline{s'}g) = \bar{r}z$. If $s \neq s'$, then $s = p$ and $s' = q$; or $s = q$ and $s' = p$. In the former case

$$[\varphi(w)] = [\varphi(f)\ rr\ y\ \bar{r}v\bar{r}\ \varphi(g)] = [\varphi(f)rv\bar{r}\varphi(g)] \neq 1$$

by our observation above and in the latter

$$[\varphi(w)] = [\varphi(f)\ rur\ y\ \bar{r}\bar{r}\ \varphi(g)] = [\varphi(f)ru\bar{r}\varphi(g)] \neq 1,$$

again by our observation. Hence $s = s'$. This means $[w] = [fsy\bar{s}g] = [fg]$ and also $1 = [\varphi(w)] = [\varphi(fg)]$ and since $|fg| < |w|$, induction yields $[w] = [fg] = 1$.

Hence, we have shown that $[\varphi(w)] = 1$ implies $[w] = 1$. Since, on the other hand, $[u] = [v]$ implies $[\varphi(u)] = [\varphi(v)]$ for all $u, v \in (\{p, \bar{p}, q, \bar{q}\} \cup X)^*$, we can lift φ to a morphism witnessing $\mathbb{B}_p * \mathbb{B}_q * M \hookrightarrow \mathbb{B}_r * M$. □

As a partial converse to Proposition 5.3, we have the following. It was first shown in [20]. Since valence automata over $\mathbb{B}*\mathbb{B}$ are essentially pushdown automata and since $\mathsf{Alg}(\mathsf{VA}(\mathbf{1})) = \mathsf{Alg}(\mathsf{Reg}) = \mathsf{CF}$, the equality $\mathsf{VA}(\mathbb{B}*\mathbb{B}*M) = \mathsf{Alg}(\mathsf{VA}(M))$ generalizes the equivalence between pushdown automata and context-free grammars.

Proposition 5.6 ([20]). *For every monoid M, $\mathsf{VA}(\mathbb{B} * \mathbb{B} * M) = \mathsf{Alg}(\mathsf{VA}(M))$. Moreover, if $R_1(M) \neq \{1\}$, then $\mathsf{VA}(\mathbb{B} * M) = \mathsf{Alg}(\mathsf{VA}(M))$.*

Proof. It suffices to prove the first statement: If $R_1(M) \neq \{1\}$, then Lemma 5.5 implies $\mathsf{VA}(\mathbb{B} * M) = \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$. Observe that if $\mathcal{C}$ is a language class with $\mathcal{C} \subseteq \mathsf{CF}$, then $\mathsf{Alg}(\mathcal{C} \cup \mathsf{VA}(N)) = \mathsf{Alg}(\mathsf{VA}(N))$ for every monoid N: For each production $A \to L$ in a $(\mathcal{C} \cup \mathsf{VA}(N))$-grammar G with L from $\mathcal{C}$, we can take a context-free grammar G' generating L (with fresh non-terminals) and replace the production $A \to L$ with the productions of G'. This is yields a $\mathsf{VA}(N)$-grammar because all singleton sets are contained in $\mathsf{VA}(N)$. Therefore, since the languages in $\mathsf{VA}(\mathbb{B})$ are effectively context-free, Proposition 5.3 yields

$$\mathsf{VA}(\mathbb{B} * N) \subseteq \mathsf{Alg}(\mathsf{VA}(\mathbb{B}) \cup \mathsf{VA}(N)) = \mathsf{Alg}(\mathsf{VA}(N))$$

for every monoid N. Therefore,

$$\mathsf{VA}(\mathbb{B} * \mathbb{B} * M) \subseteq \mathsf{Alg}(\mathsf{VA}(\mathbb{B} * M)) \subseteq \mathsf{Alg}(\mathsf{Alg}(\mathsf{VA}(M))) = \mathsf{Alg}(\mathsf{VA}(M)).$$

It remains to be shown that $\mathsf{Alg}(\mathsf{VA}(M)) \subseteq \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$.

Suppose $G = (N, T, P, S)$ is a $\mathsf{VA}(M)$-grammar and let $X = N \cup T$. Since $\mathsf{VA}(M)$ is closed under union, we may assume that for each $B \in N$, there is precisely one production $B \to L_B$ in P. For each nonterminal $B \in N$, there is a valence automaton $\mathcal{A}_B = (Q_B, X, M, E_B, q_0^B, F_B)$ over M with $\mathsf{L}(\mathcal{A}_B) = L_B$. We may clearly assume that $Q_B \cap Q_C = \emptyset$ for $B \neq C$ and that for each $(p, w, m, q) \in E_B$, we have $|w| \leq 1$.

In order to simplify the correctness proof, we modify G. Let $\lfloor$ and $\rfloor$ be new symbols and let G' be the grammar $G' = (N, T \cup \{\lfloor, \rfloor\}, P', S)$, where P' consists of the productions $B \to \lfloor L \rfloor$ for $B \to L \in P$. Moreover, let

$$K = \{v \in (N \cup T \cup \{\lfloor, \rfloor\})^* \mid u \Rightarrow_{G'}^* v,\ u \in L_S\}.$$

Then $\mathsf{L}(G) = \pi_T(K \cap (T \cup \{\lfloor, \rfloor\})^*)$ and it suffices to show $K \in \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$.

Let $Q = \bigcup_{B \in N} Q_B$. For each $q \in Q$, let $\mathbb{B}_q = \langle q, \bar{q} \mid q\bar{q} = 1 \rangle$ be an isomorphic copy of $\mathbb{B}$. Let $M' = \mathbb{B}_{q_1} * \cdots * \mathbb{B}_{q_n} * M$, where $Q = \{q_1, \ldots, q_n\}$. We shall prove $K \in \mathsf{VA}(M')$, which implies $K \in \mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$ by Lemma 5.5 since $R_1(\mathbb{B} * M) \neq \{1\}$.

Let $E = \bigcup_{B \in N} E_B$, $F = \bigcup_{B \in N} F_B$. The new set E' consists of the following transitions:

(2) (p, x, m, q) for $(p, x, m, q) \in E$,

(3) $(p, \lfloor, mq, q_0^B)$ for $(p, B, m, q) \in E$, $B \in N$,

(4) $(p, \rfloor, \bar{q}, q)$ for $p \in F$, $q \in Q$.

We claim that with $\mathcal{A}' = (Q, N \cup T \cup \{\lfloor, \rfloor\}, M', E', q_0^S, F)$, we have $\mathsf{L}(\mathcal{A}') = K$.

Let $v \in K$, where $u \Rightarrow_{G'}^n v$ for some $u \in L_S$. We show $v \in \mathsf{L}(\mathcal{A}')$ by induction on n. For $n = 0$, we have $v \in L_S$ and can use transitions of type (2) inherited from $\mathcal{A}_S$ to accept v. If $n \geq 1$, let $u \Rightarrow_{G'}^{n-1} v' \Rightarrow_{G'} v$. Then $v' \in \mathsf{L}(\mathcal{A}')$ and $v' = xBy$, $v = x\lfloor w \rfloor y$ for some $B \in N$, $w \in L_B$. The run for v' uses a transition $(p, B, m, q) \in E$. Instead of using this transition, we can use $(p, \lfloor, mq, q_0^B)$, then execute the (2)-type transitions for $w \in L_B$, and finally use $(f, \rfloor, \bar{q}, q)$, where f is the final state in the run for w. This has the effect of reading $\lfloor w \rfloor$ from the input and multiplying $mq1\bar{q} = m$ to the storage monoid. Hence, the new run is valid and accepts v. Hence, $v \in \mathsf{L}(\mathcal{A}')$. This proves $K \subseteq \mathsf{L}(\mathcal{A}')$.

In order to show $\mathsf{L}(\mathcal{A}') \subseteq K$, consider the morphisms $\varphi\colon (T \cup \{\lfloor, \rfloor\})^* \to \mathbb{B}$, $\psi\colon M' \to \mathbb{B}$ with $\varphi(x) = 1$ for $x \in T$, $\varphi(\lfloor) = a$, $\varphi(\rfloor) = \bar{a}$, $\psi(q) = a$ for $q \in Q$, $\psi(\bar{q}) = \bar{a}$, and $\psi(m) = 1$ for $m \in M$. The transitions of $\mathcal{A}'$ are constructed such that $(p, \varepsilon, 1) \to_{\mathcal{A}'}^* (q, w, m)$ implies $\varphi(w) = \psi(m)$. In particular, if $v \in \mathsf{L}(\mathcal{A}')$, then $\pi_{\{\lfloor, \rfloor\}}(v)$ is a semi-Dyck word with respect to $\lfloor$ and $\rfloor$.

Let $v \in \mathsf{L}(\mathcal{A}')$ and let $n = |w|_{\lfloor}$. We show $v \in K$ by induction on n. If $n = 0$, then the run for v only used transitions of type (2) and hence $v \in L_S$. If $n \geq 1$, since $\pi_{\{\lfloor,\rfloor\}}(v)$ is a semi-Dyck word, we can write $v = x\lfloor w\rfloor y$ for some $w \in (N \cup T)^*$. Since $\lfloor$ and $\rfloor$ can only be produced by transitions of the form (3) and (4), respectively, the run for v has to be of the form

$$\begin{aligned}(q_0^S, \varepsilon, 1) &\to_{\mathcal{A}'}^* (p, x, r)\\ &\to_{\mathcal{A}'} (q_0^B, x\lfloor, rmq)\\ &\to_{\mathcal{A}'}^* (f, x\lfloor w, rmqs)\\ &\to_{\mathcal{A}'} (q', x\lfloor w\rfloor, rmqs\overline{q'})\\ &\to_{\mathcal{A}'}^* (f', x\lfloor w\rfloor y, rmqs\overline{q'}t)\end{aligned}$$

for some $p, q, q' \in Q$, $B \in N$, $(p, B, m, q) \in E$, $f, f' \in F$, $r, t \in M'$, and $s \in M$ and with $rmqs\overline{q'}t = 1$. This last condition implies $s = 1$ and $q = q'$, which in turn entails $rmt = 1$. This also means $(p, B, m, q') = (p, B, m, q) \in E$ and $(q_0^B, \varepsilon, 1) \to_{\mathcal{A}'}^* (f, w, s) = (f, w, 1)$ and hence $w \in L_B$. Using the transition $(p, B, m, q') \in E$, we have

$$\begin{aligned}(q_0^S, \varepsilon, 1) &\to_{\mathcal{A}'}^* (p, x, r)\\ &\to_{\mathcal{A}'} (q', xB, rm)\\ &\to_{\mathcal{A}'}^* (f', xBy, rmt).\end{aligned}$$

Hence $xBy \in \mathsf{L}(\mathcal{A}')$ and $|xBy|_{\lfloor} < |v|_{\lfloor}$. Thus, induction yields $xBy \in K$ and since $xBy \Rightarrow_{G'} x\lfloor w\rfloor y$, we have $v = x\lfloor w\rfloor y \in K$. This proves $\mathsf{L}(\mathcal{A}') = K$. □

For two language classes $\mathcal{C}$ and $\mathcal{D}$, we will consider the languages obtained by intersecting a language from $\mathcal{C}$ with a language in $\mathcal{D}$. Since the class of these intersections might not be well-behaved, we use a slight extension. By $\mathcal{C} \sqcap \mathcal{D}$, we denote the class of all languages $h(K \cap L)$ where $K \subseteq X^*$ belongs to $\mathcal{C}$ and $L \subseteq X^*$ is a member of $\mathcal{D}$ and $h\colon X^* \to Y^*$ is a morphism. This allows us to state the following characterization of $\mathsf{VA}(M \times N)$ in terms of $\mathsf{VA}(M)$ and $\mathsf{VA}(N)$ by Kambites [10].

Proposition 5.7. *If M, N are monoids, then $\mathsf{VA}(M \times N) = \mathsf{VA}(M) \sqcap \mathsf{VA}(N)$.*

This implies in particular that if $\mathsf{VA}(M_i) \subseteq \mathsf{VA}(N_i)$ for $i \in \{0, 1\}$, then we also have the inclusion $\mathsf{VA}(M_0 \times M_1) \subseteq \mathsf{VA}(N_0 \times N_1)$. Of course, this also means that if $\mathsf{VA}(M_i) = \mathsf{VA}(N_i)$ for $i \in \{0, 1\}$, then $\mathsf{VA}(M_0 \times M_1) = \mathsf{VA}(N_0 \times N_1)$. We are now ready to prove Proposition 3.2.

Proposition 3.2. By definition, we have $\mathbb{M}\Gamma \cong \mathbb{B} \times (M_0 * M_1)$, where $M_i \cong \mathbb{B}$ or $M_i \cong \mathbb{Z}$ for $i \in \{0, 1\}$. We show that $\mathsf{VA}(M_0 * M_1) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$ in any case. This suffices, since it clearly implies $\mathsf{VA}(\mathbb{M}\Gamma) = \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{B})$ according to Proposition 5.7. If $M_0 \cong M_1 \cong \mathbb{B}$, the equality $\mathsf{VA}(M_0 * M_1) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$ is trivial, so we may assume $M_0 \cong \mathbb{Z}$.

If $M_1 \cong \mathbb{Z}$, then $M_0 * M_1 \cong \mathbb{Z} * \mathbb{Z}$, meaning that $\mathsf{VA}(M_0 * M_1)$ is the class of context-free languages (see Eq. (1)) and thus $\mathsf{VA}(M_0 * M_1) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$.

If $M_1 \cong \mathbb{B}$, then $\mathsf{VA}(\mathbb{Z} * \mathbb{B}) = \mathsf{Alg}(\mathsf{VA}(\mathbb{Z}))$ by Proposition 5.6. Since $\mathsf{VA}(\mathbb{Z})$ is included in the context-free languages, we have $\mathsf{Alg}(\mathsf{VA}(\mathbb{Z})) = \mathsf{VA}(\mathbb{B} * \mathbb{B})$. □

We shall now prove Theorem 3.3. Note that the implication "1 ⇒ 2" immediately follows from Theorem 3.1. The implication "2 ⇒ 3" is an old graph-theoretic result of Wolk.

Theorem 5.8 (Wolk [18])**.** *A simple graph is a transitive forest if and only if it does not contain* C4 *or* P4 *as an induced subgraph.*

The implication "3 ⇒ 4" is a simple combinatorial observation. An analogous fact is part of Lohrey and Steinberg's proof of Theorem 3.4.

Lemma 5.9. *If Γ is a PPN-free transitive forest, then $\mathbb{M}\Gamma \in \mathsf{DEC}$.*

Proof. Let $\Gamma = (V, E)$. We proceed by induction on $|V|$. If Γ is empty, then $\mathbb{M}\Gamma \cong \mathbf{1} \cong \mathbb{B}^0 \in \mathsf{DEC}$. Hence, we assume that Γ is non-empty. If Γ is not connected, then Γ is the disjoint union of PPN-free transitive forests Γ_1, Γ_2, for which $\mathbb{M}\Gamma_1, \mathbb{M}\Gamma_2 \in \mathsf{DEC}$ by induction. Hence, $\mathbb{M}\Gamma \cong \mathbb{M}\Gamma_1 * \mathbb{M}\Gamma_2 \in \mathsf{DEC}$.

Suppose Γ is connected. Since Γ is a transitive forest, there is a vertex $v \in V$ such that $\Gamma \setminus v$ is a PPN-free transitive forest and v is adjacent to every vertex in $V \setminus \{v\}$. We distinguish two cases.

- If v is a looped vertex, then $\mathbb{M}\Gamma \cong \mathbb{Z} \times \mathbb{M}(\Gamma \setminus v)$, and $\mathbb{M}(\Gamma \setminus v) \in \mathsf{DEC}$ by induction.
- If v is an unlooped vertex, then Γ being PPN-free means that in $\Gamma \setminus v$, any two distinct vertices are adjacent. Hence, $\mathbb{M}\Gamma \cong \mathbb{B}^m \times \mathbb{Z}^n$, where m and n are the number of unlooped and looped vertices in Γ, respectively. Therefore, $\mathbb{M}\Gamma \in \mathsf{DEC}$.

□

For establishing Theorem 3.3, our remaining task is to prove the implication "4 ⇒ 1". In light of Theorems 3.1 and 5.8 and Lemma 5.9, this amounts to showing that emptiness is decidable for valence automata over monoids in DEC. This will involve two facts (Theorem 5.10 and Proposition 5.11) about the languages arising from monoids in DEC.

The following generalization of Parikh's theorem by van Leeuwen will allow us to exploit our description of free products by algebraic extensions. If X is an alphabet, $X^{\oplus}$ denotes the set of maps $\alpha \colon X \to \mathbb{N}$. The elements of $X^{\oplus}$ are called *multisets*. The *Parikh map* is the map $\Psi \colon X^* \to X^{\oplus}$ where $\Psi(w)(x)$ is the number of occurrences of x in w. By $\mathcal{P}(S)$, we denote the power set of the set S. A *substitution* is a map $\sigma \colon X \to \mathcal{P}(Y^*)$, where X and Y are alphabets. Given $L \subseteq X^*$, we write $\sigma(L)$ for the set of all words $v_1 \cdots v_n$, where $v_i \in \sigma(x_i)$, $1 \leq i \leq n$, for $x_1 \cdots x_n \in L$ and $x_1, \ldots, x_n \in X$. If $\sigma(x)$ belongs to $\mathcal{C}$ for each $x \in X$, then σ is a *$\mathcal{C}$-substitution*. The class $\mathcal{C}$ is said to be *substitution closed* if $\sigma(L) \in \mathcal{C}$ for every member L of $\mathcal{C}$ and every $\mathcal{C}$-substitution σ.

Theorem 5.10 (van Leeuwen [17])**.** *For each substitution closed full trio $\mathcal{C}$, we have $\Psi(\mathsf{Alg}(\mathcal{C})) = \Psi(\mathcal{C})$.*

For $\alpha, \beta \in X^{\oplus}$, let $\alpha + \beta \in X^{\oplus}$ be defined by $(\alpha + \beta)(x) = \alpha(x) + \beta(x)$. With this operation, $X^{\oplus}$ is a monoid. For a subset $S \subseteq X^{\oplus}$, we write $S^{\oplus}$ for the smallest submonoid of $X^{\oplus}$ containing S. A subset of the form $\mu + F^{\oplus}$ for $\mu \in X^{\oplus}$ and a finite $F \subseteq X^{\oplus}$ is called *linear*. A finite union of linear sets is called *semilinear*. By $\mathsf{SLI}(\mathcal{C})$ we denote the class of languages $h(L \cap \Psi^{-1}(S))$, where $h \colon X^* \to Y^*$ is a morphism, L belongs to $\mathcal{C}$, and $S \subseteq X^{\oplus}$ is semilinear.

Proposition 5.11 ([20])**.** *For each monoid M, we have*

$$\mathsf{SLI}(\mathsf{VA}(M)) = \bigcup_{i \geq 0} \mathsf{VA}(M \times \mathbb{Z}^i).$$

We will prove decidability for DEC by reducing the problem to the reachability problem of priority multicounter machines, whose decidability has been established by Reinhardt [15]. Priority multicounter machines are an extension of Petri nets with one inhibitor arc. Intuitively, a priority multicounter machine is a partially blind multicounter machine with the additional capability of restricted zero tests: The counters are numbered from 1 to k and for each $\ell \in \{1, \ldots, k\}$, there is a zero test instruction that checks whether counters 1 through ℓ are zero. Let us define priority multicounter machines formally.

A *priority k-counter machine* is a tuple $\mathcal{A} = (Q, X, E, q_0, F)$, where (i) X is an alphabet, (ii) Q is a finite set of states, (iii) E is a finite subset of $Q \times X^* \times \{0, \ldots, k\} \times \mathbb{Z}^k \times Q$, and its elements are called *edges* or *transitions*, (iv) $q_0 \in Q$ is the *initial state*, and (v) $F \subseteq Q$ is the set of *final states*. For $\ell \in \{0, \ldots, k\}$, let

$$\mathbb{N}^k_\ell = \{(\mu_1, \ldots, \mu_k) \in \mathbb{N}^k \mid \mu_1 = \ldots = \mu_\ell = 0\}.$$

We are now ready to defines the semantics of priority counter machines. A *configuration* of $\mathcal{A}$ is a pair $(q, \mu) \in Q \times \mathbb{N}^k$. For configurations (q, μ) and (q', μ'), we write $(q, \mu) \xrightarrow{w}_{\mathcal{A}} (q', \mu')$ if there are $(q_0, \mu_0), \ldots, (q_n, \mu_n) \in Q \times \mathbb{N}^k$ such that

(i) $(q, \mu) = (q_0, \mu_0)$ and $(q', \mu') = (q_n, \mu_n)$,
(ii) for each $i \in \{1, \ldots, n\}$, there is a transition $(q_{i-1}, w_i, \ell, \nu, q_i) \in E$ such that $\mu_{i-1} \in \mathbb{N}^k_\ell$ and $\mu_i = \mu_{i-1} + \nu$, and $w = w_1 \cdots w_n$.

The *language accepted by* $\mathcal{A}$ is defined as

$$\mathsf{L}(\mathcal{A}) = \{w \in X^* \mid (q_0, 0) \xrightarrow{w}_{\mathcal{A}} (f, 0) \text{ for some } f \in F\}.$$

A *priority multicounter machine* is a priority k-counter machine for some $k \in \mathbb{N}$. The class of languages accepted by priority multicounter machines is denoted by Prio. Reinhardt has shown that the reachability problem for priority multicounter machines is decidable [15], which can be reformulated as follows.

Theorem 5.12 (Reinhardt [15])**.** *The emptiness problem is decidable for priority multicounter machines.*

Although the decidability proof for the emptiness problem for valence automata over $\mathsf{SC}^{\pm}$ employs a reduction to priority multicounter machines, it should be stressed that the mechanisms realized by $\mathsf{SC}^{\pm}$ are quite different from priority counters and very likely not subsumed by them in terms of accepted languages. For example, $\mathsf{SC}^{\pm}$ contains pushdown stacks ($\mathbb{B} * \mathbb{B}$)—if the priority multicounter machines could accept all context-free languages (or even just the semi-Dyck language D_2^*), this would easily imply decidability of the emptiness problem for pushdown Petri nets. Indeed, $\mathsf{SC}^{\pm}$ can even realize stacks where each entry consists of n partially blind counters (since $\mathbb{B} * (\mathbb{B}^n) \in \mathsf{SC}^{\pm}$). On the other hand, priority multicounter machines do not seem to be subsumed by $\mathsf{SC}^{\pm}$ either: After building stacks once, $\mathsf{SC}^{\pm}$ only allows adding *blind* counters (and building stacks again). It therefore seems unlikely that a mechanism in $\mathsf{SC}^{\pm}$ can accept the languages even of a priority 2-counter machine.

The idea of the proof of "4 $\Rightarrow$ 1" is, given a valence automaton over some $M \in \mathsf{DEC}$, to construct a Parikh-equivalent priority multicounter machine. This construction makes use of the following simple fact. A full trio $\mathcal{C}$ is said to be *Presburger closed* if $\mathsf{SLI}(\mathcal{C}) \subseteq \mathcal{C}$.

Lemma 5.13. Prio *is a Presburger closed full trio and closed under substitutions.*

Proof. The fact that Prio is a full trio can be shown by standard automata constructions. Given a priority multicounter machine $\mathcal{A}$ and a semilinear set $S \subseteq X^{\oplus}$, we add $|X|$ counters to $\mathcal{A}$ that ensure that the input is contained in $\mathsf{L}(\mathcal{A}) \cap \Psi^{-1}(S)$. This proves that Prio is Presburger closed.

Suppose $\sigma \colon X \to \mathcal{P}(Y^*)$ is a Prio-substitution. Furthermore, let $\mathcal{A}$ be a priority k-counter machine and let $\sigma(x)$ be given by a priority ℓ-counter machine for each $x \in X$. We construct a priority $(\ell+k)$-counter machine $\mathcal{B}$ from $\mathcal{A}$ by adding ℓ counters. $\mathcal{B}$ simulates $\mathcal{A}$ on counters $\ell + 1, \ldots, \ell + k$. Whenever $\mathcal{A}$ reads x, $\mathcal{B}$ uses the first ℓ counters to simulate the priority ℓ-counter machine for $\sigma(x)$. Using the zero test on the first ℓ counters, it makes sure that the machine for $\sigma(x)$ indeed ends up in a final configuration. Then clearly $\mathsf{L}(\mathcal{B}) = \sigma(\mathsf{L}(\mathcal{A}))$. □

Lemma 5.14. *We have the effective inclusion* $\Psi(\mathsf{VA}(\mathsf{DEC})) \subseteq \Psi(\mathsf{Prio})$*. More precisely, given* $M \in \mathsf{DEC}$ *and* $L \in \mathsf{VA}(M)$*, one can construct an* $L' \in \mathsf{Prio}$ *with* $\Psi(L') = \Psi(L)$*.*

Proof. We proceed by induction with respect to the definition of DEC. In the case $M = \mathbb{B}^n$, we have $\mathsf{VA}(M) \subseteq \mathsf{Prio}$, because priority multicounter machines generalize partially blind multicounter machines.

Suppose $M = N \times \mathbb{Z}$ and $\Psi(\mathsf{VA}(N)) \subseteq \Psi(\mathsf{Prio})$ and let $L \in \mathsf{VA}(M)$. By Proposition 5.11, we have $L = h(K \cap \Psi^{-1}(S))$ for some semilinear set S, a morphism h, and $K \in \mathsf{VA}(N)$. Hence, there is a $\bar{K} \in \mathsf{Prio}$ with $\Psi(\bar{K}) = \Psi(K)$. With this, we have $\Psi(L) =$

$\Psi\left(h(\bar{K} \cap \Psi^{-1}(S))\right)$ and since Prio is Presburger closed, we have $h(\bar{K} \cap \Psi^{-1}(S)) \in \mathsf{Prio}$ and thus $\Psi(L) \in \Psi(\mathsf{Prio})$.

Suppose $M = M_0 * M_1$ and $\Psi(\mathsf{VA}(M_i)) \subseteq \Psi(\mathsf{Prio})$ for $i \in \{0, 1\}$. Let L be a member of $\mathsf{VA}(M)$. According to Proposition 5.3, this means L belongs to $\mathsf{Alg}(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1))$. Since $\Psi(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1)) \subseteq \Psi(\mathsf{Prio})$, we can construct a Prio-grammar G with $\Psi(\mathsf{L}(G)) = \Psi(L)$. By Theorem 5.10 and Lemma 5.13, this implies $\Psi(L) \in \Psi(\mathsf{Prio})$. □

The following lemma is a direct consequence of Lemma 5.14 and Theorem 5.12: Given a valence automaton over M with $M \in \mathsf{DEC}$, we construct a priority multicounter machine accepting a Parikh-equivalent language. The latter can then be checked for emptiness.

Lemma 5.15. *For each $M \in \mathsf{DEC}$, the emptiness problem for valence automata over M is decidable.*

This completes the proof of "4 ⇒ 1" of Theorem 3.3 and hence concludes the proof of Theorem 3.3.

6. Expressive equivalences

We now turn to the proof of Propositions 3.5 and 3.6, which characterize the expressiveness of valence automata over $\mathsf{SC}^{\pm}$ and REM, respectively.

Proposition 3.5. Since $\mathsf{SC}^{\pm} \subseteq \mathsf{DEC}$, the inclusion "$\supseteq$" is immediate. We show by induction with respect to the definition of DEC that for each $M \in \mathsf{DEC}$, there is an $M' \in \mathsf{SC}^{\pm}$ with $\mathsf{VA}(M) \subseteq \mathsf{VA}(M')$. This is trivial if $M = \mathbb{B}^n$, so suppose $\mathsf{VA}(M) \subseteq \mathsf{VA}(M')$ and $\mathsf{VA}(N) \subseteq \mathsf{VA}(N')$ for $M, N \in \mathsf{DEC}$ and $M', N' \in \mathsf{SC}^{\pm}$. Observe that by induction on the definition of $\mathsf{SC}^{\pm}$, one can show that there is a common $P \in \mathsf{SC}^{\pm}$ with $\mathsf{VA}(M') \subseteq \mathsf{VA}(P)$ and $\mathsf{VA}(N') \subseteq \mathsf{VA}(P)$. Of course, we may assume that $R_1(P) \neq \{1\}$. Then we have

$$\begin{aligned}\mathsf{VA}(M * N) &\subseteq \mathsf{Alg}(\mathsf{VA}(M) \cup \mathsf{VA}(N)) \subseteq \mathsf{Alg}(\mathsf{VA}(M') \cup \mathsf{VA}(N')) \\ &\subseteq \mathsf{Alg}(\mathsf{VA}(P)) = \mathsf{VA}(\mathbb{B} * P),\end{aligned}$$

in which the first inclusion is due to Proposition 5.3 and the equality in the end is provided by Proposition 5.6. Since $\mathbb{B} * P \in \mathsf{SC}^{\pm}$, this completes the proof for $M * N$. Moreover, $\mathsf{VA}(M) \subseteq \mathsf{VA}(M')$ implies $\mathsf{VA}(M \times \mathbb{Z}) \subseteq \mathsf{VA}(M' \times \mathbb{Z})$ and we have $M' \times \mathbb{Z} \in \mathsf{SC}^{\pm}$. □

Proposition 3.6. By induction, it is easy to see that each $M \in \mathsf{SC}^{+}$ is isomorphic to some $\mathbb{M}\Gamma$ where Γ contains a PPN-graph and Γ^{-} is a transitive forest. By Theorem 5.8, this means Γ^{-} contains neither $\mathsf{C4}$ nor $\mathsf{P4}$. This proves the inclusion "$\supseteq$".

Because of Theorem 5.8, for the inclusion "$\subseteq$", it suffices to show that if Γ^{-} is a transitive forest, then there is some $M \in \mathsf{SC}^{+}$ with $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(M)$. We prove this by induction on the number of vertices in $\Gamma = (V, E)$. As in the proof of Lemma 5.9, we may assume that for every induced proper subgraph Δ of Γ, we find an $M \in \mathsf{SC}^{+}$ with $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(M)$. If Γ is empty, then $\mathbb{M}\Gamma \cong \mathbf{1}$ and $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(\mathbb{B}^{(2)} \times \mathbb{B})$. Hence, we may assume that Γ is non-empty.

If Γ is not connected, then $\Gamma = \Gamma_1 \uplus \Gamma_2$ with non-empty graphs Γ_1, Γ_2. This implies that there are $M_1, M_2 \in \mathsf{SC}^{+}$ with $\mathsf{VA}(\mathbb{M}\Gamma_i) \subseteq \mathsf{VA}(M_i)$ for $i \in \{1, 2\}$. By induction with respect to the definition of SC^{+}, one can show that there is a common $N \in \mathsf{SC}^{+}$ with $\mathsf{VA}(M_i) \subseteq \mathsf{VA}(N)$ for $i \in \{1, 2\}$. Here, N can clearly be chosen with $R_1(N) \neq \{1\}$. Then, we have

$$\begin{aligned}\mathsf{VA}(\mathbb{M}\Gamma) = \mathsf{VA}(\mathbb{M}\Gamma_1 * \mathbb{M}\Gamma_2) &\subseteq \mathsf{Alg}(\mathsf{VA}(\mathbb{M}\Gamma_1) \cup \mathsf{VA}(\mathbb{M}\Gamma_2)) \\ &\subseteq \mathsf{Alg}(\mathsf{VA}(M_1) \cup \mathsf{VA}(M_2)) \subseteq \mathsf{Alg}(\mathsf{VA}(N)) = \mathsf{VA}(\mathbb{B} * N)\end{aligned}$$

and $\mathbb{B} * N \in \mathsf{SC}^{+}$ as in the proof of Proposition 3.5.

Suppose Γ is connected. Since Γ^{-} is a transitive forest, there is a vertex $v \in V$ that is adjacent to every vertex in $V \setminus \{v\}$. By induction, there is an $M \in \mathsf{SC}^{+}$ with $\mathsf{VA}(\mathbb{M}(\Gamma \setminus v)) \subseteq \mathsf{VA}(M)$. Depending on whether v is looped or not, we have $\mathbb{M}\Gamma \cong \mathbb{M}(\Gamma \setminus v) \times \mathbb{Z}$ or

$\mathbb{M}\Gamma \cong \mathbb{M}(\Gamma \setminus v) \times \mathbb{B}$. Since $\mathsf{VA}(\mathbb{Z}) \subseteq \mathsf{VA}(\mathbb{B} \times \mathbb{B})$ (one blind counter can easily be simulated by two partially blind counters), this yields $\mathsf{VA}(\mathbb{M}\Gamma) \subseteq \mathsf{VA}(\mathbb{M}(\Gamma \setminus v) \times \mathbb{B} \times \mathbb{B}) \subseteq \mathsf{VA}(M \times \mathbb{B} \times \mathbb{B})$ and the fact that $M \times \mathbb{B} \times \mathbb{B} \in \mathsf{SC}^+$ completes the proof. □

7. Finite-index languages and Petri nets

We have seen in the previous sections that valence automata over $\mathsf{SC}^{\pm}$ constitute a model that (strictly) subsumes Petri nets and has a decidable emptiness problem. Moreover, they feature a type of pushdown stack. A similar result has been obtained by Atig and Ganty [2]. They proved that given a finite-index context-free language and a Petri net language, it is decidable whether their intersection is empty. Here, we present a common generalization of these facts: We provide a language class that contains the languages considered by Atig and Ganty and those in $\mathsf{VA}(\mathsf{SC}^{\pm})$ and enjoys decidability of the emptiness problem.

Definitions. If $\mathcal{C}$ is the class of finite languages, a $\mathcal{C}$-grammar is also called a *context-free grammar*. For a context-free grammar $G = (N, T, P, S)$, we may assume that for each production (A, M), the set M is a singleton and instead of $(A, \{w\})$, we write $A \to w$ for the production. The grammar is said to be in *Chomsky normal form (CNF)* if for every production $A \to w$, we have $w \in N^2 \cup T \cup \{\varepsilon\}$.

The finite-index restriction considered by Atig and Ganty places a budget constraint on the nonterminal occurrences in sentential forms. This leads to a restricted derivation relation. Suppose G is in CNF. For $u, v \in (N \cup T)^*$, we write $u \Rightarrow_{G,k} v$ if $|u|_N, |v|_N \leq k$ and $u \Rightarrow_G v$. Then, the *k-approximation* $\mathsf{L}_k(G)$ of $\mathsf{L}(G)$ is defined as

$$\mathsf{L}_k(G) = \{w \in T^* \mid S \Rightarrow_{G,k}^* w\}.$$

For $k \geq 1$, we use CF_k to denote the class of languages of the form $\mathsf{L}_k(G)$ for context-free grammars G. The languages in CF_k are called *index-k context-free languages*. It will later be convenient to let CF_0 denote the regular languages. Moreover, fiCF is the union $\bigcup_{k \geq 1} \mathsf{CF}_k$. Its members are called *finite-index context-free languages*. Note that although clearly $\mathsf{L}(G) = \bigcup_{k \geq 1} \mathsf{L}_k(G)$ for every individual grammar G, the class fiCF is strictly contained in CF: Salomaa has shown that D_1^*, the semi-Dyck language over one pair of parentheses, is not contained in fiCF [16].

A *d-dimensional (labeled) Petri net*[3] is a tuple $N = (X, E, \mu_0, F)$, where X is an alphabet, T is a finite subset of $(X \cup \{\varepsilon\}) \times \mathbb{Z}^d$ whose elements are called *transitions*, $\mu_0 \in \mathbb{N}^d$ is the *initial marking*, and $F \subseteq \mathbb{N}^d$ is a finite set of *final markings*. For $\mu, \mu' \in \mathbb{N}^d$ and $w \in X^*$, we write $\mu \xrightarrow{w}_N \mu'$ if there are $\mu_0, \ldots, \mu_n \in \mathbb{N}^d$ and transitions $(x_1, \nu_1), \ldots, (x_n, \nu_n) \in T$ such that $w = x_1 \cdots x_n$, $\mu_0 = \mu$, $\mu_n = \mu'$, and $\mu_i = \mu_{i-1} + \nu_i$ for $i \in \{1, \ldots, n\}$. Moreover, we define

$$\mathsf{L}(N, \mu, \mu') = \{w \in X^* \mid \mu \xrightarrow{w}_N \mu'\}, \qquad \mathsf{L}(N) = \bigcup_{\mu \in F} \mathsf{L}(N, \mu_0, \mu).$$

The language $\mathsf{L}(N)$ is said to be *generated by* N. A languages is a *Petri net language* if it is generated by some labeled Petri net. By P, we denote the class of all Petri net languages. Observe that we have $\mathsf{P} = \bigcup_{n \geq 0} \mathsf{VA}(\mathbb{B}^n)$.

The main result of this section involves the language class $\mathsf{fiCF} \sqcap \mathsf{P}$. We will use the fact that this is a full trio, which follows from the following classical result. See [7, Theorem 3.6.1] for a proof.

Proposition 7.1. *If $\mathcal{C}$ and $\mathcal{D}$ are full trios, then $\mathcal{C} \sqcap \mathcal{D}$ is a full trio as well.*

In our notation, the result of Atig and Ganty can be stated as follows.

Theorem 7.2 (Atig and Ganty [2])**.** *The class $\mathsf{fiCF} \sqcap \mathsf{P}$ has a decidable emptiness problem.*

[3]This definition is closer to what is known as a Vector Addition System, but these models are well-known to be equivalent with respect to generated languages.

Here, we present a language class including both $\mathsf{fiCF} \sqcap \mathsf{P}$ and $\mathsf{VA}(\mathsf{SC}^{\pm})$ where emptiness is still decidable. First, consider the following hierarchy. Let

$$\mathsf{F}_0 = \mathsf{P}, \qquad \mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{Alg}(\mathsf{F}_i)) \text{ for } i \geq 0, \qquad \mathsf{F} = \bigcup_{i \geq 0} \mathsf{F}_i.$$

The class F captures the expressive power of valence automata over monoids in $\mathsf{SC}^{\pm}$:

Proposition 7.3. $\mathsf{VA}(\mathsf{SC}^{\pm}) = \mathsf{F}$.

Proof. For the inclusion "$\subseteq$", we prove that for every $M \in \mathsf{SC}^{\pm}$, we have $\mathsf{VA}(M) \subseteq \mathsf{F}_i$ for some $i \geq 0$. Clearly, we have $\mathsf{VA}(\mathbb{B}^n) \subseteq \mathsf{F}_0$. Moreover, if $\mathsf{VA}(M) \subseteq \mathsf{F}_i$, then

$$\mathsf{VA}(M \times \mathbb{Z}) \subseteq \mathsf{SLI}(\mathsf{VA}(M)) \subseteq \mathsf{SLI}(\mathsf{F}_i) \subseteq \mathsf{F}_{i+1},$$

in which the first inclusion follows from Proposition 5.11. Finally, if $\mathsf{VA}(M_0) \subseteq \mathsf{F}_i$ and $\mathsf{VA}(M_1) \subseteq \mathsf{F}_j$, then $\mathsf{VA}(M_0), \mathsf{VA}(M_1) \subseteq \mathsf{F}_k$ for $k = \max\{i, j\}$ and thus

$$\mathsf{VA}(M_0 * M_1) \subseteq \mathsf{Alg}(\mathsf{VA}(M_0) \cup \mathsf{VA}(M_1)) \subseteq \mathsf{Alg}(\mathsf{F}_k) \subseteq \mathsf{F}_{k+1},$$

where the first inclusion is due to Proposition 5.3. This completes the proof of the inclusion "$\subseteq$".

For the inclusion "$\supseteq$", we show by induction on i that $\mathsf{F}_i \subseteq \mathsf{VA}(\mathsf{DEC})$ for every $i \geq 0$. Since $\mathsf{VA}(\mathsf{DEC}) = \mathsf{VA}(\mathsf{SC}^{\pm})$ by Proposition 3.5, this is sufficient. Clearly, the inclusion $\mathsf{F}_0 = \bigcup_{n \geq 0} \mathsf{VA}(\mathbb{B}^n) \subseteq \mathsf{VA}(\mathsf{DEC})$ holds. Now suppose $\mathsf{F}_i \subseteq \mathsf{VA}(\mathsf{DEC})$ and let L be a member of $\mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{Alg}(\mathsf{F}_i))$. This means we have $L = h(K \cap \Psi^{-1}(S))$ for some homomorphism h, a language K from $\mathsf{Alg}(\mathsf{F}_i)$, and a semilinear set S. As a member of $\mathsf{Alg}(\mathsf{F}_i)$, the language K is generated by an F_i-grammar G. Each right-hand side in G is contained in F_i and thus, by induction, in $\mathsf{VA}(\mathsf{DEC})$. Hence, suppose the right-hand sides of G are $K_1, \ldots, K_n$ with $K_i \in \mathsf{VA}(M_i)$ for $M_1, \ldots, M_n \in \mathsf{DEC}$. Consider the monoid $M = M_1 * \cdots * M_n$. Since each M_i embeds into M, the languages $K_1, \ldots, K_n$ belong to $\mathsf{VA}(M)$. Thus, K is a member of $\mathsf{Alg}(\mathsf{VA}(M))$, which equals $\mathsf{VA}(\mathbb{B} * \mathbb{B} * M)$ according to Proposition 5.6. According to Proposition 5.11, this implies that L belongs to $\mathsf{VA}((\mathbb{B} * \mathbb{B} * M) \times \mathbb{Z}^k)$ for some $k \geq 0$. Since $(\mathbb{B} * \mathbb{B} * M) \times \mathbb{Z}^k$ is a member of DEC, we know that L belongs to $\mathsf{VA}(\mathsf{DEC})$. We have thus shown $\mathsf{F}_{i+1} \subseteq \mathsf{VA}(\mathsf{DEC})$, which establishes the inclusion "$\subseteq$". □

Our new class is defined as follows. Let

$$\mathsf{G}_0 = \mathsf{fiCF} \sqcap \mathsf{P}, \qquad \mathsf{G}_{i+1} = \mathsf{SLI}(\mathsf{Alg}(\mathsf{G}_i)) \text{ for } i \geq 0, \qquad \mathsf{G} = \bigcup_{i \geq 0} \mathsf{G}_i.$$

Then clearly $\mathsf{F}_i \subseteq \mathsf{G}_i$ for $i \geq 0$ and hence $\mathsf{VA}(\mathsf{SC}^{\pm}) = \mathsf{F} \subseteq \mathsf{G}$. Moreover, we obviously have $\mathsf{fiCF} \sqcap \mathsf{P} \subseteq \mathsf{G}$. We shall prove the following.

Theorem 7.4. *The class* G *has a decidable emptiness problem.*

We show Theorem 7.4 by proving a slightly stronger version of Theorem 7.2: Atig and Ganty reduce the emptiness problem of $\mathsf{fiCF} \sqcap \mathsf{P}$ to the emptiness problem for priority multicounter machines. We strengthen this slightly and show that for each language L in $\mathsf{fiCF} \sqcap \mathsf{P}$, one can construct a priority multicounter machine $\mathcal{A}$ with $\Psi(\mathsf{L}(\mathcal{A})) = \Psi(L)$, in other words: $\Psi(\mathsf{fiCF} \sqcap \mathsf{P}) \subseteq \Psi(\mathsf{Prio})$. This allows us to apply Theorem 5.10 and Lemma 5.13 to conclude that $\Psi(\mathsf{G}) \subseteq \Psi(\mathsf{Prio})$.

The following observation provides a decomposition of languages in fiCF. A context-free grammar $G = (N, T, P, S)$ is called *linear* if every production $A \to w$ in G satisfies $|w|_N \leq 1$. A language is called *linear context-free* if it is generated by a linear context-free grammar. Note that CF_1 is precisely the class of linear context-free languages.

Proposition 7.5. *Suppose* $k \geq 1$. *A language belongs to* CF_k *if and only if it can be written as* $\sigma(L)$ *for a linear context-free language* L *and a* CF_{k-1}*-substitution* σ.

Proof. We prove the statement by induction on k. For $k = 1$, it essentially states that linear context-free languages are closed under regular substitutions, which is clearly true. For the induction step, we use a result of Atig and Ganty. Let $G = (N, T, P, S)$ be a context-free grammar in CNF. For each $i \geq 0$, let $A^{[i]}$ be a fresh symbol. For each $\ell \geq 0$, we define a grammar $G^{[\ell]}$ as follows. We have $G^{[\ell]} = (N^{[\ell]}, T, P^{[\ell]}, S^{[\ell]})$ with $N^{[\ell]} = \{A^{[i]} \mid A \in N,\ 0 \leq i \leq \ell\}$ and $P^{[\ell]}$ is the smallest set of productions such that

(1) for each $A \to BC$ in P, we have $A^{[i]} \to B^{[i]}C^{[i-1]}$ and $A^{[i]} \to B^{[i-1]}C^{[i]}$ in $P^{[\ell]}$ for every index $i \in \{1, \ldots, \ell\}$,
(2) for each $A \to w$ in P with $w \in T \cup \{\varepsilon\}$, we have $A^{[i]} \to w$ in $P^{[\ell]}$ for every $i \in \{0, \ldots, \ell\}$.

For each nonterminal A of G, we define

$$\mathsf{L}(G, A) = \{w \in T^* \mid A \Rightarrow_G^* w\}, \qquad \mathsf{L}_\ell(G, A) = \{w \in T^* \mid A \Rightarrow_{G,\ell}^* w\}.$$

Atig and Ganty [2] show that for every $i \in \{0, \ldots, \ell\}$, one has

$$\mathsf{L}(G^{[\ell]}, A^{[i]}) = \mathsf{L}_{i+1}(G, A). \tag{5}$$

Now suppose K belongs to CF_{k+1} with $K = \mathsf{L}_{k+1}(G)$ where G is in CNF. According to (5), we have $\mathsf{L}(G^{[k]}) = K$. We now construct a (linear) context-free grammar $G' = (N', T', P', S')$ (that is not necessarily in CNF) as follows. It has terminal symbols $T' = T \cup \{A^{[i]} \mid 0 \leq i \leq k-1\}$ and its nonterminal symbols are $N' = \{A^{[k]} \mid A \in N\}$. As productions, it contains all those productions of G whose left-hand side belongs to N'. Moreover, the substitution $\sigma\colon T'^* \to \mathcal{P}(T^*)$ is defined as follows: For $a \in T$, we set $\sigma(a) = \{a\}$. For $A^{[i]} \in T'$, we define $\sigma(A^{[i]}) = \mathsf{L}(G^{[k]}, A^{[i]})$. Since for $A^{[i]} \in T'$, we have $i \leq k-1$, the equation (5) tells us that $\sigma(A^{[i]})$ belongs to $\mathsf{CF}_{i+1} \subseteq \mathsf{CF}_k$. Hence, σ is a CF_k-substitution. Moreover, an inspection of the definition of $G^{[\ell]}$ yields that G' is clearly linear. Finally, we have $K = \sigma(\mathsf{L}(G'))$, so that with $L = \mathsf{L}(G')$, we have proven the "only if" direction of the proposition. The other direction is obvious. □

We now turn to the key lemma (Lemma 7.6) of our slightly stronger version of Atig and Ganty's result. We want to show that given an index-k context-free language K and a Petri net language L, one can construct a priority multicounter machine $\mathcal{A}$ with $\Psi(\mathsf{L}(\mathcal{A})) = \Psi(K \cap L)$. The proof proceeds by induction on the index k, which warrants a strengthening of the statement.

We need some terminology. For a vector $\mu = (m_1, \ldots, m_d) \in \mathbb{N}^d$ and $k \geq d$, we denote by $0|\mu$ the vector $(0, \ldots, 0, m_1, \ldots, m_d) \in \mathbb{N}^k$. The dimension k will always be clear from the context. In order to make the induction work, we need to construct priority counter machines with the additional property that for a particular d, they never zero-test their d topmost counters. Therefore, for a priority k-counter machine $\mathcal{A} = (Q, X, E, q_0, F)$ and $d \leq k$, we define $\mathcal{A}_d$ to be the machine obtained from $\mathcal{A}$ removing all transitions (q, x, ℓ, ν, q') with $\ell > d$. In other words, we remove all transitions that perform a zero-test on a counter other than $1, \ldots, d$. We define the language

$$\mathsf{L}_d(\mathcal{A}, q, \mu, q', \mu') = \{w \in X^* \mid (q, 0|\mu) \xrightarrow{w}_{\mathcal{A}_d} (q', 0|\mu')\}.$$

For a language $K \subseteq X^*$ and a d-dimensional Petri net $N = (X, E, \mu_0, F)$, we say that a priority k-counter machine $\mathcal{A}$ is a (K, N)-*simulator* if $k \geq d$ and there are two states p and p' in $\mathcal{A}$ such that for every $\mu, \mu' \in \mathbb{N}^d$, we have

$$\Psi(\mathsf{L}_d(\mathcal{A}, p, \mu, p', \mu')) = \Psi(K \cap \mathsf{L}(N, \mu, \mu')). \tag{6}$$

In this case, p and p' are called *source* and *target*, respectively.

Lemma 7.6. *Given a language K in* fiCF *and a labeled Petri net N, one can construct a (K, N)-simulator.*

Proof. Let $N = (X, E, \mu_0, F)$ be a d-dimensional Petri net and let K belong to CF_k. We proceed by induction on k. If $k = 0$, then K is accepted by some finite automaton $\mathcal{B}$. We may assume that $\mathcal{B}$ has an initial state p and one final state p'. One can construct a priority d-counter machine $\mathcal{A}$ by a product construction from N and $\mathcal{B}$ such that $\mathcal{A}$ has the same state set as $\mathcal{B}$ and

$$\mathsf{L}_d(\mathcal{A}, p, \mu, p', \mu') = K \cap \mathsf{L}(N, \mu, \mu'),$$

meaning it is indeed a (K, N)-simulator.

For the induction step, suppose $k \geq 1$. According to Proposition 7.5, there is a linear context-free language $L \subseteq Y^*$ and a CF_{k-1}-substitution $\sigma \colon Y \to \mathcal{P}(X^*)$ with $K = \sigma(L)$. Let us begin with some explanation. Since L is linear context-free, it is given by a grammar $G = (\bar{N}, Y, P, S)$ where every production is of the form $A \to x_1 B x_2$ or $A \to \varepsilon$ with $A, B \in \bar{N}$ and $x_1, x_2 \in Y \cup \{\varepsilon\}$. Let $D \subseteq P^*$ be the regular language of production sequences that correspond to derivations in G and let $g_1, g_2 \colon P^* \to Y^*$ be the morphisms where for $\pi = A \to x_1 B x_2$ (with $x_1, x_2 \in Y \cup \{\varepsilon\}$), we set $g_i(\pi) = x_i$. Then, we have $L = \{g_1(w) g_2(w^R) \mid w \in D\}$.

Therefore, if τ_i is the CF_{k-1}-substitution with $\tau_i(\pi) = \sigma(g_i(\pi))$ for $i = 1, 2$, then $K = \sigma(L)$ consists of all words in $\tau_1(w)\tau_2(w^R)$ for $w \in D$. In other words, K contains precisely those words of the form

$$u_1 \cdots u_n v_n \cdots v_1 \tag{7}$$

such that there is a word $w = \pi_1 \cdots \pi_n \in D$, $\pi_1, \ldots, \pi_n \in P$ with $u_i \in \tau_1(\pi_i)$ and $v_i \in \tau_2(\pi_i)$ for $i \in \{1, \ldots, n\}$.

Our task is to construct a (K, N)-simulator $\mathcal{A}'$. This means, using a priority counter machine, we have to simulate—up to Parikh image—all runs of N with labels as in Eq. (7). By induction, we have a $(\tau_i(\pi), N)$-simulator $\mathcal{A}_{\pi,i}$ for each $\pi \in P$ and $i \in \{1, 2\}$. Each of the machines $\mathcal{A}_{\pi,i}$ has $\geq d$ counters, so we may clearly assume that for some $\ell \geq 0$, they all have $\ell + d$ counters. Moreover, by definition of a $(\tau_i(\pi), N)$-simulator, these machines never perform a zero-test on the d top-most counters. Moreover, using a zero-test, we can guarantee that when $\mathcal{A}_{\pi,i}$ reaches its target state, its first ℓ counters are zero.

The basic idea is that $\mathcal{A}'$ performs a run of an automaton for D, which reads a word $w = \pi_1 \cdots \pi_n$. For each $j = 1, \ldots, n$, it executes a computation of $\mathcal{A}_{\pi_j,1}$ (reading u_j) and a computation of $\mathcal{A}_{\pi_j,2}$ (reading v_j). Hence, $\mathcal{A}'$ reads the word $u_1 v_1 u_2 v_2 \cdots u_n v_n$, which is clearly Parikh-equivalent to $u_1 \cdots u_n v_n \cdots v_1$.

We have to make sure that all these runs of the machines $\mathcal{A}_{\pi_j,i}$ are compatible in the sense that they can be executed in the order prescribed by Eq. (7). To this end, all the executions of $\mathcal{A}_{\pi_1,1}, \ldots, \mathcal{A}_{\pi_n,1}$ share one set of $\ell + d$ counters. The executions of $\mathcal{A}_{\pi_1,2}, \ldots, \mathcal{A}_{\pi_n,2}$ also share a set of counters, but they are executed backwards. The counters for the backward execution are also $\ell + d$ many, but since each execution of some $\mathcal{A}_{\pi,i}$ leaves the first ℓ counters empty, the forward and the backward simulation can share the first ℓ counters between them. This leaves us with $\ell + 2d$ counters: We use counters $1, \ldots, \ell + d$ to simulate $\mathcal{A}_{\pi_j,1}$ and we use counters $1, \ldots, \ell$ and $\ell + d + 1, \ldots, \ell + 2d$ to simulate $\mathcal{A}_{\pi_j,2}$ (backwards). Therefore, we call counters $1, \ldots, \ell$ *auxiliary counters*, whereas the counters $\ell + 1, \ldots, \ell + d$ are called *forward counters*. The counters $\ell + d + 1, \ldots, \ell + 2d$ are dubbed *backward counters*.

In addition, we have to make sure that the executions of N corresponding to $v_n \cdots v_1$ can be executed after the executions corresponding to $u_1 \cdots u_n$. Therefore, after executing the run of the automaton for D, $\mathcal{A}'$ simultaneously counts down the forward and the backward counters and then performs a zero-test on the counters $1, \ldots, \ell + 2d$.

Finally, in order to be a (K, N)-simulator, $\mathcal{A}'$ must have d top-most counters so that the following holds: If we simulate the computation $\mu \xrightarrow{u_1 \cdots u_n v_n \cdots v_1}_N \mu'$ with $\mu, \mu' \in \mathbb{N}^d$, then the d top-most counters of $\mathcal{A}'$ must contain μ in the beginning and μ' in the end. To this end, we add an additional set of d counters, called *global counters*. Hence, in total, $\mathcal{A}'$ has

$\ell + 3d$ counters:

$$\underbrace{1, \ldots, \ell}_{\text{auxiliary}}, \underbrace{\ell+1, \ldots, \ell+d}_{\text{forward}}, \underbrace{\ell+d+1, \ldots, \ell+2d}_{\text{backward}}, \underbrace{\ell+2d+1, \ldots, \ell+3d}_{\text{global}}.$$

The global counters are used as follows. The machine $\mathcal{A}'$ starts with counters $0|\mu \in \mathbb{N}^{\ell+3d}$. First, it nondeterministically subtracts some vector $\nu_1 \in \mathbb{N}^d$ from the global counters and simultaneously adds it to the forward counters. Then, it nondeterministically adds a vector $\nu_2' \in \mathbb{N}^d$ to both the global counters and the backward counters. After performing the simulation of the $\mathcal{A}_{\pi_1,1}, \ldots, \mathcal{A}_{\pi_n,1}$ and the $\mathcal{A}_{\pi_1,2}, \ldots, \mathcal{A}_{\pi_n,2}$, suppose the forward counters contain $\nu_1' \in \mathbb{N}^d$ and the backward counters contain $\nu_2 \in \mathbb{N}^d$. As described above, $\mathcal{A}'$ afterwards compares the forward and backward counters, ensuring that $\nu_1' = \nu_2$ and thus:

$$\nu_1 \xrightarrow{u_1 \cdots u_n}_N \nu_1' = \nu_2 \xrightarrow{v_n \cdots v_1}_N \nu_2'.$$

Observe that this guarantees that the global counters of $\mathcal{A}'$ reflect the counters of the simulated computation of N: In the end, they are precisely $0|\mu' \in \mathbb{N}^{\ell+3d}$, where $\mu' = \mu - \nu_1 + \nu_2'$, which means $\mu \xrightarrow{u_1 \cdots u_n v_n \cdots v_1}_N \mu'$.

Let us make the description of $\mathcal{A}'$ more precise.

(i) $\mathcal{A}'$ has a state p, where it nondeterministically subtracts tokens from the global counters and simultaneously adds them to the forward counters.

(ii) Note that $D \subseteq P^*$ can be accepted by a finite automaton with state set $\bar{N}$, the non-terminals of G. Therefore, from the state p, $\mathcal{A}'$ can enter the state $S \in \bar{N}$ to start simulating the automaton for D.

(iii) In a state $A \in \bar{N}$, $\mathcal{A}'$ selects a production $\pi = A \to x_1 B x_2 \in P$ and then executes a computation of $\mathcal{A}_{\pi,1}$ in the auxiliary and the forward counters. Then, it executes a computation of $\mathcal{A}_{\pi,2}$ backwards on the auxiliary and backward counters. Then, it switches to state $B \in \bar{N}$.

(iv) If $\mathcal{A}'$ is in state $A \in \bar{N}$ and there is a production $A \to \varepsilon \in P$, then $\mathcal{A}'$ switches to a state p'', in which it simultaneously counts down the forward and the backward counters. From p'' it non-deterministically switches to p' while performing a zero-test on all counters $1, \ldots, \ell + 2d$.

In conclusion, it is clear that for $\mu, \mu' \in \mathbb{N}^d$, we have $(p, 0|\mu) \xrightarrow{w}_{\mathcal{A}'} (p', 0|\mu')$ if and only if $w = u_1 v_1 u_2 v_2 \cdots u_n v_n$ such that there is a word $\pi_1 \cdots \pi_n \in D$, $\pi_1, \ldots, \pi_n \in P$, such that $u_j \in \tau_1(\pi_j)$ and $v_j \in \tau_2(\pi_j)$. Thus, $\mathcal{A}'$ is a (K, N)-simulator.

□

We are now ready to prove the slightly stronger version of the decidability result of Atig and Ganty.

Theorem 7.7. *Given K in $\mathsf{fiCF} \sqcap \mathsf{P}$, one can construct a priority multicounter machine $\mathcal{A}$ with $\Psi(\mathsf{L}(\mathcal{A})) = \Psi(K)$.*

Proof. Since the languages of priority multicounter machines are closed under morphisms, we may assume that $K = C \cap P$, where C is in CF_k and $P = \mathsf{L}(N)$ for a d-dimensional labeled Petri net $N = (X, T, \mu_0, F)$. Lemma 7.6 allows us to construct a (C, N)-simulator $\mathcal{A}$ with source p and target p'. This means, for each $\mu \in \mathbb{N}^d$, we have $\Psi(\mathsf{L}_d(\mathcal{A}, p, \mu_0, p', \mu)) = \Psi(C \cap \mathsf{L}(N, \mu_0, \mu))$ and in particular

$$\Psi\left(\underbrace{\bigcup_{\mu \in F} \mathsf{L}_d(\mathcal{A}, p, \mu_0, p', \mu)}_{=:L}\right) = \Psi\left(\bigcup_{\mu \in F} C \cap \mathsf{L}(N, \mu_0, \mu)\right) = \Psi(C \cap P).$$

Since we can clearly construct a priority multicounter machine for L, the proof of the theorem is complete. □

This allows us to prove Theorem 7.4.

Theorem 7.4. Given a language in G_i, we can recursively construct a Parikh equivalent priority multicounter machine. According to Theorem 7.7, this is true of $\mathsf{G}_0 = \mathsf{fiCF} \sqcap \mathsf{P}$. Furthermore, Theorem 5.10 and Lemma 5.13 tell us that if we can carry out such a construction for G_i, we can also do it for G_{i+1}. □

8. Conclusion

Of course, an intriguing open question is whether the storage mechanisms corresponding to SC^+ have a decidable reachability problem. First, since their simplest instance are pushdown Petri nets, this extends the open question concerning the latter's reachability. Second, they naturally subsume the priority multicounter machines of Reinhardt. This makes them a candidate for being a quite powerful model for which reachability might be decidable.

Observe that if these storage mechanisms turn out to exhibit decidability, this would mean that the characterization of Lohrey and Steinberg (Theorem 3.4) remains true for all graph monoids. This can be interpreted as evidence for decidability.

Acknowledgments. The author is grateful to the anonymous referees of both the conference and the journal version. Their helpful comments have greatly improved the presentation of this work.

References

[1] I. J. Aalbersberg and H. J. Hoogeboom. Characterizations of the decidability of some problems for regular trace languages. *Mathematical Systems Theory*, 22(1):1–19, 1989.

[2] M. F. Atig and P. Ganty. Approximating Petri net reachability along context-free traces. In *Proceedings of the 31st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 152–163, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[3] J. Berstel. *Transductions and Context-Free Languages*. Teubner, 1979.

[4] P. Buckheister and G. Zetzsche. Semilinearity and context-freeness of languages accepted by valence automata. In K. Chatterjee and J. Sgall, editors, *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS 2013)*, volume 8087 of *Lecture Notes in Computer Science*, pages 231–242, Berlin/Heidelberg, 2013. Springer-Verlag.

[5] V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.

[6] H. Fernau and R. Stiebe. Sequential grammars and automata with valences. *Theoretical Computer Science*, 276:377–405, 2002.

[7] S. Ginsburg. *Algebraic and automata-theoretic properties of formal languages*. North-Holland Publishing Company Amsterdam, 1975.

[8] S. Ginsburg and S. Greibach. Principal AFL. *Journal of Computer and System Sciences*, 4(4):308–338, 1970.

[9] J. Hartmanis and J. E. Hopcroft. What makes some language theory problems undecidable. *Journal of Computer and System Sciences*, 4(4):368–376, 1970.

[10] M. Kambites. Formal languages and groups as memory. *Communications in Algebra*, 37:193–208, 2009.

[11] M. Kambites, P. V. Silva, and B. Steinberg. On the rational subset problem for groups. *Journal of Algebra*, 309:622–639, 2007.

[12] J. Leroux, G. Sutre, and P. Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP 2015)*, pages 324–336, Berlin/Heidelberg, 2015. Springer-Verlag.

[13] M. Lohrey. The rational subset membership problem for groups: a survey. In C. M. Campbell, M. R. Quick, E. F. Robertson, and C. M. Roney-Dougal, editors, *Groups St Andrews 2013*, volume 422 of *London Mathematical Society Lecture Note Series*, pages 368–389, Cambridge, United Kingdom, 2015. Cambridge University Press.

[14] M. Lohrey and B. Steinberg. The submonoid and rational subset membership problems for graph groups. *Journal of Algebra*, 320(2):728–755, 2008.

[15] K. Reinhardt. Reachability in Petri nets with inhibitor arcs. *Electronic Notes in Theoretical Computer Science*, 223:239–264, 2008. Proceedings of the Second Workshop on Reachability Problems in Computational Models (RP 2008).

[16] A. Salomaa. On the index of a context-free grammar and language. *Information and Control*, 14(5):474–477, 1969.

[17] J. van Leeuwen. A generalisation of Parikh's theorem in formal language theory. In *Proceedings of the 2nd International Colloquium on Automata, Languages and Programming (ICALP 1974)*, volume 14 of *Lecture Notes in Computer Science*, pages 17–26, Berlin/Heidelberg, 1974. Springer-Verlag.

[18] E. S. Wolk. A note on "the comparability graph of a tree". *Proceedings of the American Mathematical Society*, 16(1):17–20, 1965.

[19] G. Zetzsche. Silent transitions in automata with storage. In *Proc. of the 40th International Colloquium on Automata, Languages and Programming (ICALP 2013)*, volume 7966 of *LNCS*, pages 434–445, Berlin Heidelberg, 2013. Springer.

[20] G. Zetzsche. Computing downward closures for stacked counter automata. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 743–756, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[21] G. Zetzsche. The emptiness problem for valence automata or: Another decidable extension of Petri nets. In *Proceedings of the 9th International Workshop on Reachability Problems (RP 2015)*, volume 9328 of *Lecture Notes in Computer Science*, pages 166–178, Berlin/Heidelberg, 2015. Springer-Verlag.

[22] G. Zetzsche. Monoids as storage mechanisms. *Bulletin of the EATCS*, 120:237–249, 2016.

[23] G. Zetzsche. *Monoids as Storage Mechanisms*. PhD thesis, Technische Universität Kaiserslautern, 2016.

Email address: zetzsche@lsv.fr

LSV, CNRS & ENS Paris-Saclay, France