Monoids as Storage Mechanisms

Georg Zetzsche
Technische Universität Kaiserslautern,
Germany

An important theme of automata theory is to study mathematical models of computing devices with regard to what behavior they can exhibit and what we can infer about such a device when given a description.

These two types of questions each have their own motivation. The first type addresses *expressiveness*. This aspect is important to understand because it explains what we can compute with limited resources and what systems we can describe with the respective models. The second type of questions explores the *analyzability* of models. This perspective is instrumental when we want to algorithmically verify properties of systems, which, due to the advent of increasingly complex and concurrent systems, has become a task of significant importance.

The perspectives of expressiveness and analyzability are deeply intertwined: They are conflicting qualities insofar as the more expressive a model is, the more difficult it usually is to analyze. For these reasons, it has become a strong driving force of today's research in theoretical computer science to understand how we can provide models that are expressive enough for a given type of systems and yet are simple enough to be amenable to analysis. This thesis contributes by studying the relationship between the computational properties of automata with storage and the employed storage mechanism.

Automata with storage In a tradition initiated by Turing in the introduction of the eponymous machine, automata theory yielded a rich variety of models that comprise a finite-state control and a potentially infinite data repository. The models are obtained by imposing restrictions on how the data can be stored, manipulated, and retrieved, while permitting arbitrary use of the finite-state control.

In terms of hard- and software systems that can be represented by such models, this means we can precisely reflect arbitrary control flows, but we abstract from certain aspects of data access. For example, pushdown automata can correctly imitate the control flow and calling stack of a recursive program, but heap memory cannot be represented. A form of data respository, together with the permitted modes of access, is called a *storage mechanism*. Examples of storage mechanisms include Turing machine tapes, pushdown storages, and various kinds of counters.

Instead of investigating the properties of an individual model of computation, the present work attempts to provide general insights into how expressiveness and analyzability of a model of computation are affected by the structure of the storage mechanism. To this end, it presents generalizations of results about concrete storage mechanisms to larger classes of storage mechanisms. These generalizations will characterize those storage mechanisms for which the given result remains true and for which it fails.

Storage mechanisms as monoids In order to speak of classes of storage mechanisms, we need an overarching framework that accommodates each of the concrete storage mechanisms we wish to address. Such a framework is provided by interpreting storage mechanisms as *monoids*.

Suppose a storage mechanism consists of a (potentially infinite) set of states, a finite set of functions representing its available operations, an initial state, and a collection of valid final states. To account for operations that are not always applicable, such as a pop operation for a stack symbol that is not currently at the top, the functions can be partial functions. For example, a pushdown storage with stack alphabet Γ consists of the set Γ^* as its set of states, the operations push_a and pop_a for each $a \in \Gamma$, and the empty word ε as its initial state and its final state (assuming that it accepts with an empty stack). As partial functions, the operations push_a and pop_a are defined as

$$\mathsf{push}_a \colon \Gamma^* \to \Gamma^*, \quad w \mapsto wa \qquad\qquad \mathsf{pop}_a \colon \Gamma^* \nrightarrow \Gamma^*, \quad wa \mapsto w.$$

(here, we denote partial functions by $\nrightarrow$). Note that pop_a is defined on precisely those words that end in a.

Another example is the Minsky counter, which has $\mathbb{N}$, the set of natural numbers, as its set of states and has inc (*increment*), dec (*decrement*), and zero (*zero test*) as its operations:

$$\mathsf{inc} \colon \mathbb{N} \to \mathbb{N}, \quad n \mapsto n+1, \qquad \mathsf{dec} \colon \mathbb{N} \nrightarrow \mathbb{N}, \quad n \mapsto n-1, \qquad \mathsf{zero} \colon \mathbb{N} \nrightarrow \mathbb{N}, \quad 0 \mapsto 0.$$

Note that here, the decrement operation is undefined for state 0 and the zero test operation is defined only in state 0.

To such a storage mechanism, we can associate the monoid of all compositions of available operations. Let us examine what this yields in the case of a pushdown store as above. If we compose push_a and pop_b for $a \neq b$, we obtain the function 0, which is defined nowhere: After pushing an a, popping b cannot be defined. Moreover, composing 0 with any other operation yields 0 again. If, however, we

only consider compositions where such incompatible push and pop do not occur, the reader can verify that we always get functions of the form $P_{u,v}$ for $u, v \in \Gamma^*$, where

$$\begin{aligned} P_{u,v} \colon \Gamma^* &\twoheadrightarrow \Gamma^*, \\ wu &\mapsto wv, \end{aligned}$$

is defined on precisely those words with suffix u. Therefore, the resulting monoid has the elements $\{0\} \cup \{P_{u,v} \mid u, v \in \Gamma^*\}$.

Let us consider the case of a Minsky counter. Any composition of just the increment and decrement operations yields an element $C_{r,s}$ such that

$$\begin{aligned} C_{r,s} \colon \mathbb{N} &\twoheadrightarrow \mathbb{N}, \\ n + r &\mapsto n + s, \end{aligned}$$

which is defined on all numbers $\geq r$. If the composition involves a zero test, then it is either 0 as above or it is defined on only one element $r \in \mathbb{N}$ and of the form $D_{r,s}$, for which

$$\begin{aligned} D_{r,s} \colon \mathbb{N} &\twoheadrightarrow \mathbb{N}, \\ r &\mapsto s. \end{aligned}$$

Hence, the corresponding monoid comprises the set $\{0, C_{r,s}, D_{r,s} \mid r, s \in \mathbb{N}\}$.

Monoids as storage mechanisms The advantage of interpreting storage mechanisms as monoids is that we can go in the other direction and interpret *monoids as storage mechanisms*: The elements of the monoid determine the set of states as well as the set of operations and the identity element is the final state. This allows us to use algebraic constructions to synthesize similar storage mechanisms and thus identify *what structural traits of the mechanism are responsible for which computational properties*. For example, we will define monoids by graphs that may contain self-loops. We will then see that graphs with no self-loops or edges correspond to pushdown storages. If the graph has no self-loops, but is otherwise a clique, it is equivalent to counters without zero tests (that cannot go below zero). This is usually called a set of *partially blind counters*. Moreover, if the graph is a clique and has self-loops everywhere, we obtain counters that can go below zero and are only zero tested in the end of the computation, hence a collection of *blind counters*.

This means we can regard these individual storage as points on a spectrum and examine where exactly the computational properties remain true and where they cease to hold. For example, it is known that automata with a pushdown or with blind counters accept languages with semilinear Parikh images, which is not true of partially blind counters. We can now study which graphs exactly guarantee semilinearity of the accepted languages.

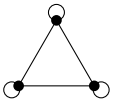
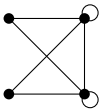
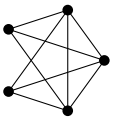
Graph Γ	Monoid $\mathbb{M}\Gamma$	Storage mechanism
	$\mathbb{B}^{(3)}$	Pushdown (with three symbols)
	$\mathbb{B}^3$	Three partially blind counters
	$\mathbb{Z}^3$	Three blind counters
	$\mathbb{B}^{(2)} \times \mathbb{Z}^2$	Pushdown (with two symbols) and two blind counters
	$\mathbb{B}^{(2)} \times \mathbb{B}^3$	Pushdown and three partially blind counters
	$\mathbb{B}^{(2)} \times \mathbb{B}^{(2)}$	Two pushdowns (with two symbols each)

Table 1: Examples of storage mechanisms

Valence automata We investigate monoids as storage mechanisms by deploying them in the framework of valence automata. A valence automaton over a monoid M is a finite automaton in which each edge carries an input word and an element of M. Let us define this formally. Let M be a monoid. A *valence automaton over* M is a tuple $A = (Q, X, M, E, q_0, F)$, where

- Q is a finite set of *states*,
- X is an alphabet, called its *input alphabet*,
- $E \subseteq Q \times X^* \times M \times Q$ is a finite set of *edges* or *transitions*,
- $q_0 \in Q$ is its *initial state*, and

- $F \subseteq Q$ is its set of *final states*.

For triples (q, u, x) and (q', u', x') from $Q \times X^* \times M$, let

$$(q, u, x) \to_A (q', u', x') \qquad \text{if } u' = uw \text{ and } x' = xz \text{ for some } (q, w, z, q') \in E.$$

In this work, we measure the expressive power of valence automata by the class of languages they can accept. The *language accepted by A* is defined as

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

In other words, the automaton accepts all words that label paths from an initial state to a final state such that the product of the monoid elements is the identity.

Valence automata are not a new concept and have been studied before by several authors from various perspectives. What distinguishes this work from earlier ones is that it systematically generalizes results for individual models of automata with storage. Specifically, for each of a series of results about concrete storage mechanisms, it presents a broader class of monoids and identifies those members of the class to which the result carries over.

As a rich source of monoids to represent well-known storage mechanims, this work also introduces *graph monoids*. They are defined by graphs, which often allows us to relate the combinatorial properties of the graphs with the computational properties of the resulting storage mechanisms.

Main contributions

The following are the main contributions of this work.

Graph monoids Oftentimes, characterizing *all* monoids with a given computational property is infeasible; or it would result in a mere reformulation of the property and thus not be meaningful.

Therefore, it is useful to have a class of monoids that is large enough to accomodate storage mechanisms from the literature, but small enough to permit meaningful characterizations. To this end, this work introduces the class of *graph monoids*, which are defined by graphs. Let us sketch their definition. Suppose we have an undirected graph $\Gamma = (V, E)$ that may have self-loops. This means, $E \subseteq \{S \subseteq V \mid |S| \leq 2\}$. For brevity, we call a vertex *looped* if it has a self-loop; otherwise we call it *unlooped*. We define the alphabet $X_\Gamma = \{a_v, \bar{a}_v \mid v \in V\}$ and consider the smallest congruence $\equiv_\Gamma$ on X_Γ^* with

$$a_v \bar{a}_v \equiv_\Gamma \varepsilon \qquad \text{for each } v \in V, \text{ and} \tag{1}$$

$$xy \equiv_\Gamma yx \qquad \text{for each } x \in \{a_v, \bar{a}_v\}, \; y \in \{a_w, \bar{a}_w\} \text{ with } \{v, w\} \in E. \tag{2}$$

In other words, for each vertex $v \in V$, we have a positive element a_v and a negative element $\bar{a}_v$. Then, (1) means that $a_v\bar{a}_v$ cancels to the identity and (2) tells us that two elements (whether they are positive or negative) may commute if their vertices are adjacent. The resulting monoid is now defined as

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

It is not hard to see that by choosing appropriate graphs, one can realize pushdown automata, partially blind counter automata, blind counter automata, and Turing machines, but also various combinations thereof.

For example, suppose Γ contains just one unlooped vertex. Then $X_\Gamma = \{a_v, \bar{a}_v\}$ and $w \equiv_\Gamma \varepsilon$ if and only if $|w|_{a_v} = |w|_{\bar{a}_v}$ and $|p|_{a_v} \geq |p|_{\bar{a}_v}$ for every prefix p of w. In other words, if we interpret a_v and $\bar{a}_v$ as *increment* and *decrement*, respectively, then this means w is a sequence of counter actions that leads from 0 to 0 and keeps the counter non-negative. Since $[w] = 1$ if and only if $w \equiv_\Gamma \varepsilon$, this means $\mathbb{M}\Gamma$ represents a partially blind counter.

Furthermore, if Γ consists of one looped vertex, then $w \equiv_\Gamma \varepsilon$ if and only if $|w|_{a_v} = |w|_{\bar{a}_v}$. Hence, $\mathbb{M}\Gamma$ represents a blind counter.

Suppose Γ consists of two vertices x, y without any self-loops. If we then for $z \in \{x, y\}$, interpret a_z as a push_z operation and $\bar{a}_z$ as a pop_z operation, then $w \equiv_\Gamma \varepsilon$ if and only if w transforms the empty stack into the empty stack. Thus, $\mathbb{M}\Gamma$ realizes a pushdown store with a two-letter stack alphabet.

Moreover, if Γ is obtained from the disjoint graphs Γ_0 and Γ_1 by adding edges between any vertex from Γ_0 and any vertex from Γ_1, then $w \equiv_\Gamma \varepsilon$ if and only if $w_i \equiv_{\Gamma_i} \varepsilon$ for $i \in \{0, 1\}$, where w_i is obtained from w by deleting all symbols corresponding to letters from Γ_{1-i}. Hence, $\mathbb{M}\Gamma$ allows us to use the storage mechanisms realized by $\mathbb{M}\Gamma_0$ and $\mathbb{M}\Gamma_1$ independently.

Table 1 presents examples of graphs and the corresponding storage mechanisms. Here, the symbol $\mathbb{B}$ denotes the monoid $\mathbb{M}\Gamma$ where Γ consists of one unlooped vertex. Moreover, $M^{(n)}$ denotes the n-fold free product of the monoid M. Analogously, M^n is the n-fold direct product.

Increasing expressiveness We present an algebraic characterization of those monoids that *increase the expressiveness* in the following sense: Without the storage mechanism, finite automata only accept regular languages. Hence, we describe those monoids M for which valence automata over M can accept non-regular languages. In fact, we show that this also characterizes those monoids for which deterministic valence automata are expressively weaker and those for which valence grammars can generate non-context-free languages. Valence grammars are a concept related to valence automata and equip context-free grammars with a monoid storage. While the characterization of monoids that increase expressiveness in valence automata has been obtained independently by Render [17]

in her thesis, the latter characterization for valence grammars answers an open question raised by Fernau and Stiebe [3].

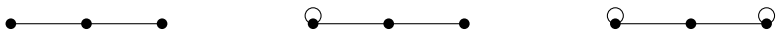
These results have been published in [20].

Emptiness problem Afterwards, we turn to the decidability of the *emptiness problem*. This is a type of reachability problem and one of the most basic problems in the algorithmic analysis of system models. Therefore, we are interested in which storage mechanisms permit a decision procedure.

There is a serious obstacle to a complete characterization: Using graph monoids, one can realize a pushdown storage with partially blind counters (see Table 1), for which the deciability of the emptiness problem remains a long-standing open question [11, 16].

However, if we forbid the subgraphs corresponding to these mechanisms, we can characterize those with a decidable emptiness problem. The result generalizes the decidability for pushdown automata and for partially blind counter automata (or equivalently, Petri nets).

The complete statement requires some terminology. A graph Γ is said to be a *PPN-graph* if it is isomorphic to one of the following three graphs:



One can show that every such storage mechanisms allows us to simulate automata with a pushdown and one partially blind counter: PPN stands for *pushdown Petri net*. A graph Γ is called *PPN-free* if it has no PPN-graph as an induced subgraph. 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 simple graph is a *transitive forest* if it is the disjoint union of comparability graphs of trees. Let Γ^- denote the graph obtained from Γ by deleting all self-loops.

Theorem 1. *Let Γ be PPN-free. Then the following conditions are equivalent:*

1. *Emptiness is decidable for valence automata over $\mathbb{M}\Gamma$.*
2. *Γ^- is a transitive forest.*

Moreover, we present an intuitive, more mechanical, description of (a) the mechanisms shown to be decidable and (b) the storage mechanisms where decidability remains open.

These mechanisms instances of *stacked counters*. Stacked counter storage mechanisms are obtained by alternating two transformations of storage mechanisms: *building stacks* (of configurations of an existing mechanism) and *adding counters* (to an existing mechanism). Building stacks works as follows: Given

one storage mechanism, we construct a new one whose configurations are stacks (i.e. sequences) of configurations of the old one. During a computation, one can then start a new entry, manipulate the topmost entry (as prescribed by the old mechanism) and pop the topmost entry if empty[1]. On the level of monoids, this corresponds to transforming M into $\mathbb{B} * M$. Adding counters is a simpler transformation: In the new mechanism, we have a counter in addition to the old mechanism. On the monoid level, this means we turn M into $\mathbb{Z} \times M$ (*adding a blind counter*) or into $\mathbb{B} \times M$ (*adding a partially blind counter*).

The mechanisms of (a) are obtained from partially blind counters by *building stacks* and *adding blind counters*. Formally, $\mathsf{SC}^{\pm}$ is the smallest class of monoids that contains $\mathbb{B}^n$ for every $n \geq 0$ and has the property that if M belongs to $\mathsf{SC}^{\pm}$, then both $\mathbb{B} * M$ and $\mathbb{Z} \times M$ belong to $\mathsf{SC}^{\pm}$ as well. Note that the monoids in $\mathsf{SC}^{\pm}$ are not precisely those satisfying the conditions of Theorem 1, but they are expressively equivalent and serve to provide an intuition.

The mechanisms of (b) are defined similarly: They are obtained from partially blind counters by *building stacks* and *adding partially blind counters*. As monoids, these mechanisms are represented by the class SC^{+}, which is the smallest class containing $\mathbb{B}^n$ for $n \geq 0$ such that if M belongs to SC^{+}, we also have $\mathbb{B} * M$ and $\mathbb{B} \times M$ in SC^{+}. Again, these mechanisms are expressively equivalent to those where Theorem 1 leaves the decidability status open.

The mechanisms corresponding to SC^{+} are a natural generalization of Reinhardt's priority counter machines but also of pushdown storages with partially blind counters. In particular, they are a promising candidate for a quite powerful model where reachability might still be decidable.

Theorem 1 extends a result of Lohrey and Steinberg [13], which characterizes those graph groups with a decidable rational subset membership problem. Where Lohrey and Steinberg rely on semilinearity arguments, we use a reduction to the reachability problem of priority multicounter machines, which has been proven decidable by Reinhardt [16].

This result has been published in [22].

Boolean closure We are also concerned with closure properties of the languages accepted by valence automata. Since it is well-known that the regular languages are closed under the Boolean operations (union, intersection, and complementation), we ask for which monoids M, the class of languages accepted by valence automata over M is *closed under the Boolean operations*.

Aside from understanding closure properties of automata models, this question

[1]Note that this is akin to (but not quite the same as) the step from order-n pushdowns to order-$(n+1)$ pushdowns. However, in contrast to higher-order pushdown automata, there is no operation to copy the topmost entry.

is relevant to the decidability of the first-order theory of structures: Identifying new monoids that admit these closure properties and decidability of the emptiness problem would yield an extended notion of automatic structures [10], whose first-order theory would be decidable.

Our result is a rather negative answer and goes beyond valence automata. It is shown here that every language class that is closed under the Boolean operations and rational transductions and contains an *arbitrary* non-regular language already includes the whole arithmetical hierarchy. The crucial idea is an encoding of counter values of a Minsky machine by Myhill-Nerode classes of the non-regular language.

It follows in particular that every language class induced by valence automata beyond the regular languages either fails to be closed under the Boolean operations or lacks virtually all decidability properties.

This result has been published in [14, 23].

Context-freeness We compare the expressiveness of storage mechanisms with that of context-free grammars. Specifically, we ask which monoids cause valence automata to only accept context-free languages. We characterize those graph products M of monoids for which valence automata over M accept only context-free languages. This means, in particular, that we extend a group-theoretic result of Lohrey and Sénizergues [12], which characterizes those graph products of groups where the resulting group is virtually free.

This result has been published in [1].

Semilinearity We study generalizations of Parikh's Theorem [15], which states that the Parikh image of each context-free language is semilinear. This result is an extraordinarily useful tool, both for proving non-expressibility result and in the algorithmic analysis of formal languages. It has been extended to so many other language classes that the term 'a Parikh theorem' has come to mean a statement guaranteeing effective semilinearity. This type of results has countless applications. Especially in cooperation with Presburger arithmetic, it facilitates a number of decision procedures.

Therefore, understanding what storage mechanisms admit a Parikh theorem is useful for clarifying expressiveness, but especially in order to analyze automata. Hence, we study which monoids guarantee semilinearity of the accepted language class. The first presented result is a characterization of those graph monoids that guarantee semilinear Parikh images. As explained above, this generalizes the semilinearity results for pushdown automata and blind multicounter automata.

A *looped clique* is a graph where every vertex is looped and any two vertices are adjacent.

Theorem 2. *Valence automata over $\mathbb{M}\Gamma$ have effectively semilinear Parikh images if and only if:*

1. *Γ^- is a transitive forest and*

2. *the neighborhood of every unlooped vertex in Γ is a looped clique.*

Moreover, we identify another type of stacked counters as expressively complete among those mechanisms with semilinearity. They are similar to the mechanisms in the results on the emptiness problem. Namely, they are obtained by alternatingly *building stacks* and *adding blind counters*. Furthermore, stacked counters exhibit a range of properties desirable for analysis and they offer a way to model recursive programs with numeric data types (Hague and Lin [6] have applied a model that is subsumed by stacked counter automata).

More precisely, SC^- is the smallest class of monoids that contains the trivial monoid $\mathbf{1}$ and has the property that if M belongs to SC^-, then both $\mathbb{B} * M$ and $\mathbb{Z} \times M$ belong to SC^- as well. To summarize, we have three types of stacked counters: (i) SC^-, where we only have blind counters, (ii) $\mathsf{SC}^\pm$, where we start with partially blind counters, but after building stacks, we can only add blind counters, and (iii) SC^+, where we start with partially blind counters and can add them even after building stacks.

These results have been published in [1].

Silent transitions For each storage mechanism, an important question is whether silent transitions (i.e. those which read no input but can manipulate the storage) are necessary to accept all languages. Indeed, if silent transitions can be eliminated, we can decide the membership status of a given input word by examining a finite number of paths through the automaton. Therefore, we ask for which monoids we can avoid silent transitions. We show that among a class of storage mechanisms, stacked counters of the type SC^- are those where this is possible. Again, this generalizes the corresponding fact for (i) pushdown automata, (ii) blind multicounter automata, and (iii) automata with access to a pushdown storage and blind counters. Results (i) and (ii) had been established by Greibach [4, 5] and (iii) is due to Hoogeboom [9].

These results have been published in [21].

Computing downward closures We also consider the computation of downward closures. It is well-known that the downward closure, i.e. the set of (not necessarily contiguous) subwords, of every language is regular [7, 8]. Moreover, computing a finite automaton for the downward closure of a given language would make a range of analysis techniques applicable. However, this cannot be done in

general. In fact, there are only few known methods for computing downward closures for languages. It is shown here that for all those graph monoids that guarantee semilinearity (equivalently, for stacked counters), downward closures can be computed. This generalizes the computability of downward closures for context-free languages, as obtained by van Leeuwen [18] and Courcelle [2].

This result has been published in [19].

Bibliography

[1] P. Buckheister and G. Zetzsche. "Semilinearity and Context-Freeness of Languages Accepted by Valence Automata". In: *Proc. of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS 2013)*. Vol. 8087. LNCS. Springer-Verlag, 2013, pp. 231–242.

[2] B. Courcelle. "On constructing obstruction sets of words". In: *Bulletin of the EATCS* 44 (1991), pp. 178–186.

[3] H. Fernau and R. Stiebe. "Sequential grammars and automata with valences". In: *Theoretical Computer Science* 276 (2002), pp. 377–405.

[4] S. A. Greibach. "Remarks on blind and partially blind one-way multicounter machines". In: *Theoretical Computer Science* 7.3 (1978), pp. 311 –324.

[5] S. A. Greibach. "A New Normal-Form Theorem for Context-Free Phrase Structure Grammars". In: *Journal of the ACM* 12.1 (1965), pp. 42–52.

[6] M. Hague and A. W. Lin. "Model Checking Recursive Programs with Numeric Data Types". In: *Proc. of the 23rd International Conference on Computer Aided Verification (CAV 2000)*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. LNCS. Berlin/Heidelberg: Springer-Verlag, 2011, pp. 743–759.

[7] L. H. Haines. "On free monoids partially ordered by embedding". In: *Journal of Combinatorial Theory* 6.1 (1969), pp. 94–98.

[8] G. Higman. "Ordering by divisibility in abstract algebras." In: *Proceedings of the London Mathematical Society. Third Series* 2 (1952), pp. 326–336.

[9] H. J. Hoogeboom. "Context-Free Valence Grammars - Revisited". In: *Proc. of the 5th International Conference on Developments in Language Theory (DLT 2001)*. Ed. by W. Kuich, G. Rozenberg, and A. Salomaa. Vol. 2295. LNCS. Berlin/Heidelberg: Springer-Verlag, 2002, pp. 293–303.

[10] B. Khoussainov and A. Nerode. “Automatic Presentations of Structures”. In: *LCC: International Workshop on Logic and Computational Complexity*. Vol. 960. LNCS. Berlin/Heidelberg: Springer-Verlag, 1995, pp. 367–392.

[11] J. Leroux, G. Sutre, and P. Totzke. “On the Coverability Problem for Pushdown Vector Addition Systems in One Dimension”. In: *Proc. of the 42nd International Colloquium on Automata, Languages and Programming (ICALP 2015)*. Berlin/Heidelberg: Springer-Verlag, 2015, pp. 324–336.

[12] M. Lohrey and G. Sénizergues. “When Is a Graph Product of Groups Virtually-Free?” In: *Communications in Algebra* 35.2 (2007), pp. 617–621.

[13] M. Lohrey and B. Steinberg. “The submonoid and rational subset membership problems for graph groups”. In: *Journal of Algebra* 320.2 (2008), pp. 728–755.

[14] M. Lohrey and G. Zetzsche. “On Boolean closed full trios and rational Kripke frames”. In: *Proc. of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*. Vol. 25. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 530–541.

[15] R. J. Parikh. “On Context-Free Languages”. In: *Journal of the ACM* 13.4 (1966), pp. 570–581.

[16] K. Reinhardt. “Reachability in Petri Nets with Inhibitor Arcs”. In: *Electronic Notes in Theoretical Computer Science* 223 (2008). Proceedings of the Second Workshop on Reachability Problems in Computational Models (RP 2008), pp. 239–264.

[17] E. Render. “Rational Monoid and Semigroup Automata”. PhD thesis. University of Manchester, 2010.

[18] J. van Leeuwen. “Effective constructions in well-partially-ordered free monoids”. In: *Discrete Mathematics* 21.3 (1978), pp. 237–252.

[19] G. Zetzsche. “Computing Downward Closures for Stacked Counter Automata”. In: *Proc. of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 743–756.

[20] G. Zetzsche. “On the Capabilities of Grammars, Automata, and Transducers Controlled by Monoids”. In: *Proc. of the 38th International Colloquium on Automata, Languages and Programming (ICALP 2011)*. Vol. 6756. LNCS. Springer-Verlag, 2011, pp. 222–233.

[21] G. Zetzsche. “Silent Transitions in Automata with Storage”. In: *Proc. of the 40th International Colloquium on Automata, Languages and Programming (ICALP 2013)*. Vol. 7966. LNCS. Springer-Verlag, 2013, pp. 434–445.

[22] G. Zetzsche. “The Emptiness Problem for Valence Automata or: Another Decidable Extension of Petri Nets”. In: *Proc. of the 9th International Workshop on Reachability Problems (RP 2015)*. Vol. 9328. LNCS. Springer-Verlag, 2015, pp. 166–178.

[23] G. Zetzsche, M. Lohrey, and D. Kuske. *On Boolean closed full trios and rational Kripke frames*. Extended version of [14]. To appear in Theory of Computing Systems. 2016.