Parikh One-Counter Automata

Michaël Cadilhac
DePaul University, Chicago, IL, USA

Arka Ghosh
University of Warsaw, Poland

Guillermo A. Pérez
University of Antwerp – Flanders Make, Belgium

Ritam Raha
University of Antwerp – Flanders Make, Belgium
LaBRI, University of Bordeaux, France

Abstract

Counting abilities in finite automata are traditionally provided by two orthogonal extensions: adding a single counter that can be tested for zeroness at any point, or adding $\mathbb{Z}$-valued counters that are tested for equality only at the end of runs. In this paper, finite automata extended with both types of counters are introduced. They are called Parikh One-Counter Automata (POCA): the "Parikh" part referring to the evaluation of counters at the end of runs, and the "One-Counter" part to the single counter that can be tested during runs.

Their expressiveness, in the deterministic and nondeterministic variants, is investigated; it is shown in particular that there are deterministic POCA languages that cannot be expressed without nondeterminism in the original models. The natural decision problems are also studied; strikingly, most of them are no harder than in the original models. A parametric version of nonemptiness is also considered.

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages

Keywords and phrases Parikh automata, Context-free languages, One-counter automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.30

Funding Belgian FWO "SAILor" project (G030020N) & NCN grant 2019/35/B/ST6/02322.

1 Introduction

Extensions of finite automata abound in the literature, with a traditional common goal: To find computational models with good expressiveness for which relevant problems are decidable. The impetus lies in formal verification: Can I express my process using that new model, then formally check that it does not have "bad" behaviors? Typically, processes are thus implemented with expressive models, while bad behaviors can be represented using a regular language. To answer the verification question, the key computational problem is then *inclusion in a regular language* (are all the executions of my process not bad?).

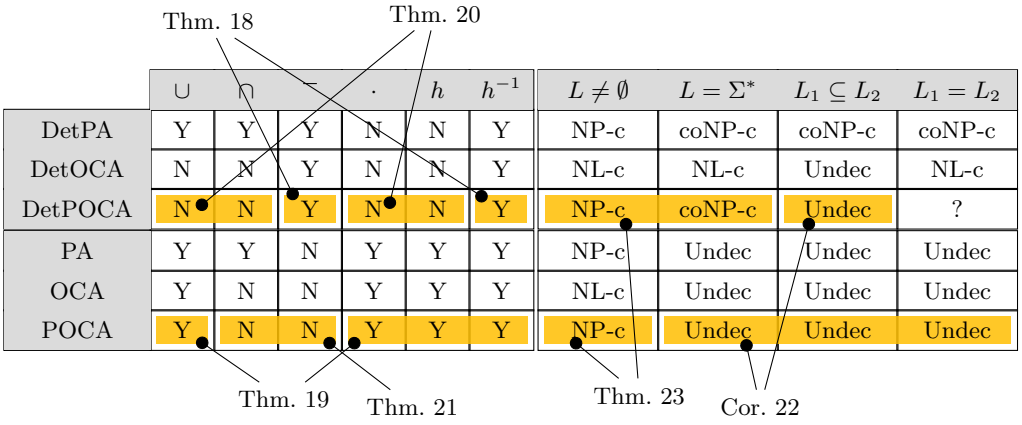
A common approach to extending finite automata is to equip them with counters or some sort of counting abilities. The literature crystallizes around two main extensions:

- Adding a *single* counter which can be tested for zeroness throughout the run. A typical language that such an extension can recognize is $L_1 = \{a^n b^n \mid n \geq 0\}^*$ (mind the star!).
- Adding any constant number of counters, but they can only be tested for zeroness or equality a bounded number of times (during the run or at the end only, these variants being equivalent for nondeterministic machines). This includes reversal-bounded counter machines [14, 6] and Parikh automata [16, 4, 5], which, incidentally, are equally expressive. A typical language in these extensions is $L_2 = \{a^n b^n c^n \mid n \geq 0\}$.

Unsurprisingly, L_1 cannot be recognized using the second extension, while L_2 cannot be recognized with the first. A natural extension is, thus, to combine these two approaches to counting into a single model, prompting the questions: **1.** Is the expressiveness of the combined model more interesting than just the union of the two original models; and **2.** Are the good decidability properties of the original models retained in the combined one?

Here, we report on this extension. The model, called *Parikh One-Counter Automata* (POCA), consists in a finite automaton with one counter which can be tested for zeroness and any number of $\mathbb{Z}$-valued counters that are checked at the end of the run using a Presburger formula (an arithmetic formula of first-order logic with addition).[1] We contribute:

- A (de)pumping lemma allowing for the study of the limits of the expressiveness of POCA. Pumping in one-counter automata is fairly simple, as it follows standard arguments for pushdown automata. In Parikh automata (i.e., POCA without the unbounded counter), any cycle taken twice can be moved around without changing membership. However, combining these two properties for POCA proves to be a great technical challenge.
- A complete picture of the relationships between POCA, one-counter automata, and Parikh automata, in their nondeterministic and deterministic variants. In addition to separation of the classes of languages under consideration, we observe that some languages that are only *nondeterministic* for both Parikh automata and one-counter automata turn out to be *deterministic* for POCA. This is of special interest in the context of verification since many problems are undecidable for nondeterministic machines but decidable for deterministic ones. We also study how the base models of Parikh automata and one-counter automata are "embedded" in POCA (Theorem 15, the statement of which should be clear at this point of the Introduction).
- A study of the decision problems for POCA and its deterministic variant. Strikingly, emptiness and inclusion in a regular language are no more complex than with Parikh automata: coNP-complete. We also study *parametric* POCA, in which parameters $x, y, \ldots$ can be used in the counter updates (as $+x$, $-y$, for instance), and show that it is undecidable whether, for all parameter values, the language of the POCA is nonempty. We relate this problem to considerations on arithmetic theories since it is one of the main motivations behind the study of parametric models [1].



	$\cup$	$\cap$	$\neg$	$\cdot$	h	h^{-1}	$L \neq \emptyset$	$L = \Sigma^*$	$L_1 \subseteq L_2$	$L_1 = L_2$
DetPA	Y	Y	Y	N	N	Y	NP-c	coNP-c	coNP-c	coNP-c
DetOCA	N	N	Y	N	N	Y	NL-c	NL-c	Undec	NL-c
DetPOCA	N	N	Y	N	N	Y	NP-c	coNP-c	Undec	?
PA	Y	Y	N	Y	Y	Y	NP-c	Undec	Undec	Undec
OCA	Y	N	N	Y	Y	Y	NL-c	Undec	Undec	Undec
POCA	Y	N	N	Y	Y	Y	NP-c	Undec	Undec	Undec

Figure 1 Closure properties and complexity results. Results about Parikh automata (PA) and one-counter automata (OCA) are from the literature [7, 16, 5, 17, 21, 2]. The left side of the table lists closure properties; h and h^{-1} mean closure under morphisms and inverse morphisms.

[1] We rely on a slightly different but equivalent definition, in which the Presburger formula actually specifies a relation on the number of times each transition is taken in the run. This explains the use of Rohit Parikh's name: a run is accepting if its Parikh image is accepted by the Presburger formula. Formal definitions appear in Section 2.

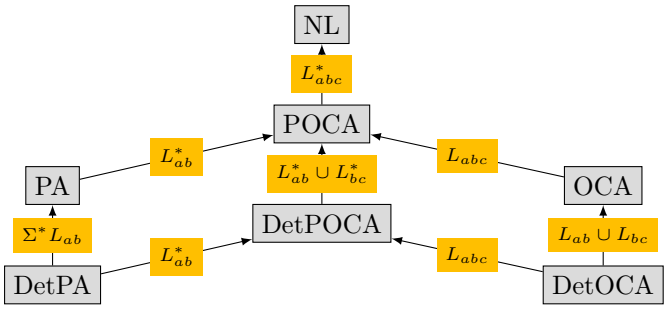


Figure 2 Separations among deterministic and nondeterministic variants of PA, OCA & POCA. The arrows denote the strict subclass relation, e.g., DetPA $\subsetneq$ PA. All the classes not having a sequence of arrows between them are incomparable. The language in between classes belongs to the higher class but not to the lower.

Organization of the paper. We recall some classical notions and introduce our models in Section 2 then present some examples of POCA in Section 3. In Section 4, we state our (de)pumping lemma for POCA and rely on it to show that some languages are not expressible. We study the relationships between our classes of languages in Section 5, the closure properties of our models in Section 6, and the complexity of decision problems in Section 7.

2 Preliminaries

We assume the reader to be familiar with elementary automata theory.

Sets. We write $\mathbb{N} = \{0, 1, 2, \ldots\}$ and $\mathbb{N}_{>0} = \{1, 2, \ldots\}$. Let $d \in \mathbb{N}_{>0}$. A set $E \subseteq \mathbb{Z}^d$ is said to be *linear* if there exist vectors $\boldsymbol{v_0}, \boldsymbol{v_1}, \ldots, \boldsymbol{v_k} \in \mathbb{Z}^d$ such that $E = \{\boldsymbol{v_0} + \sum_{i=1}^{k} x_i \boldsymbol{v_i} \mid x_1, \ldots, x_k \in \mathbb{N}\}$. A set is *semilinear* if it is a finite union of linear sets. Equivalently, a set $E \subseteq \mathbb{Z}^d$ is semilinear if it can be represented as the set of vectors satisfying a *Presburger formula* with d free variables, that is, a first-order formula over $(\mathbb{N}, +)$.

Words, languages. We usually use Σ for alphabets, write ε for the empty word, and let Σ^ε be $\Sigma \cup \{\varepsilon\}$, with the understanding that $\varepsilon \notin \Sigma$. Any alphabet in this paper is implicitly totally ordered, so that we can speak of the i-th letter of the alphabet. This is only useful in defining the *Parikh image* $\Phi(w)$ of a word $w \in \Sigma^*$: this is the vector in $\mathbb{N}^{|\Sigma|}$ whose i-th component is the number of times the i-th letter of Σ appears in w.

Given two alphabets Σ, Γ, any function $\Sigma \to \Gamma^*$ can be uniquely extended to a function $h\colon \Sigma^* \to \Gamma^*$, called a *morphism*, in such a way that $h(\varepsilon) = \varepsilon$ and $h(u \cdot v) = h(u) \cdot h(v)$. For a language $L \subseteq \Sigma^*$, we write $h(L)$ for $\{h(w) \mid w \in L\}$.

Given a language $L \subseteq \Sigma^*$, two words $u, v \in \Sigma^*$ are *Myhill-Nerode equivalent* if for any $w \in \Sigma^*$, $uw \in L \Leftrightarrow vw \in L$. This is an equivalence relation, and we write $[u]_L$ for the set of words Myhill-Nerode equivalent to u. We will be mostly interested in $[\varepsilon]_L$, the set of words that can be erased from or inserted at the beginning of any word without changing its membership to L.

Parikh one-counter automata. A *Parikh One-Counter Automaton* (POCA) $\mathcal{A}$ is a tuple $(Q, q_0, \Sigma, \Delta_0, \Delta_+, F, \varphi)$ where:

- Q is a finite set of states and $q_0 \in Q$ is the initial state,
- Σ is an alphabet,

- $\Delta_0 \subseteq Q \times \Sigma^\varepsilon \times \{0,1\} \times Q$ is a *zero-value* transition relation,
- $\Delta_+ \subseteq Q \times \Sigma^\varepsilon \times \{-1,0,1\} \times Q$ is a *positive* transition relation,
- $F \subseteq Q$ is a set of final states, and
- φ is an existential Presburger formula with $(|\Delta_0| + |\Delta_+|)$ free variables.

A *run* in $\mathcal{A}$ is a sequence of transitions:

$$\rho = (q_1, \ell_1, b_1, q_2)(q_2, \ell_2, b_2, q_3) \cdots (q_{n-1}, \ell_{n-1}, b_{n-1}, q_n) \in (\Delta_0 \uplus \Delta_+)^*.$$

We say that ρ *starts* in q_1 and *ends* in q_n. Its *trace* is the sequence of partial sums of the b_i, representing the current value of the counter:

$$\mathsf{trace}(\rho) = \left(\sum_{i<1} b_i, \quad \sum_{i<2} b_i, \quad \ldots, \quad \sum_{i<n} b_i \right),$$

with the understanding that the first term of that sequence is zero. The i-th element of the trace is simply written $\mathsf{trace}(\rho)_i$. The run ρ is:

- *counter-correct* if for all i, $\mathsf{trace}(\rho)_i = 0 \to \rho_i \in \Delta_0$ and $\mathsf{trace}(\rho)_i \neq 0 \to \rho_i \in \Delta_+$. In other words, a transition from Δ_0 is taken if the current value of the counter is 0 and one from Δ_+ if the counter is nonzero.
- *initial* if it starts in q_0, *final* if it ends in a state in F.
- *constraint-correct* if $\Phi(\rho)$ satisfies φ.
- *accepting* if it is initial, final, counter-correct, and constraint-correct.

The *label* of the run ρ is the concatenation of all the ℓ_i and a word w is accepted by $\mathcal{A}$ if it is the label of an accepting run. Finally, the *language recognized* by the POCA $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A})$ of words accepted by it.

A POCA $\mathcal{A}$ is said to be *deterministic* (DetPOCA) if for all states q and for both $\Delta \in \{\Delta_0, \Delta_+\}$:

- There are no two transitions in Δ from q having the same label; and
- If there is a transition in Δ from q labeled ε, there is no other transition from q in Δ.

Parikh Automata & One-Counter Automata. These two models are restrictions of POCA:

- A *Parikh Automaton* (PA) is a POCA in which $\Delta_0 \subseteq Q \times \Sigma^\varepsilon \times \{0\} \times Q$. In this case, Δ_+ is not useful, so we simply omit it from the tuple representation and do not write the 0 update of Δ_0. We use DetPA for the deterministic variant.
- A *One-Counter Automaton* (OCA) is a POCA in which φ is a tautology; we then omit it from the tuple representation. We use DetOCA for the deterministic variant.

3 Examples

We start with a few examples of POCA languages, which will help in clarifying the relationship of POCA with PA and OCA.

▶ **Example 1.** Let $T = \{a^n b^n \mid n > 0\}$ and define:

$$L = \{u_1 u_2 \cdots u_m c^m \mid m > 0 \wedge u_1, u_2, \ldots, u_m \in T\}.$$

This language can be shown to be unrecognizable by PA or OCA. Intuitively, it is not a PA language since it has unbounded sequences of words from T, each of which necessitating an equality check, and it is not an OCA language since m and n, respectively the number of c

and the number of a, b, require two separate counters. However, L is recognizable using a POCA. Indeed, the counter can be used to check that each subword in a^+b^+ is in T and the semilinear constraint can be used to check that the number of words u_i is exactly the number of c. In fact, this process is deterministic, hence L is a DetPOCA language.

▶ **Example 2.** Let again $T = \{a^n b^n \# \mid n > 0\}$ and $K = \{a^n b^n c^n \# \mid n > 0\}$ and define:

$$L_T = \{a, b, \#\}^* T \{a, b, \#\}^* \quad \text{and} \quad L_K = \{a, b, c, \#\}^* K \{a, b, c, \#\}^*.$$

Using a single counter, L_T can be deterministically recognized: every time we switch from b to a, we can reset the counter in search for a subword in T. Thus L_T is a DetOCA (hence DetPOCA) language. However, it can be shown that L_T is a PA language, but not a DetPA language with tools from [4]. As for L_K, since K cannot be recognized by an OCA, we need to use the semilinear constraint to recognize it: a PA for L_K would simply guess the position of the word in K, read the next $a^+b^+c^+$ using a separate part of the automaton, then check that there were as many a, b, and c in this subword using the semilinear constraint. Thus L_K is a PA (hence POCA) language. We can show, however, that it is not a DetPOCA language.

▶ **Example 3.** The previous example seems to indicate that a language that is expressible with both PA and OCA can only be deterministic if it is expressible with a DetPA or DetOCA. This is not the case. Take for instance $B = \{a^i b^j c^j \mid i \neq j \vee j \neq k\}$ and define:

$$L = \{a^n \# u_1 \# u_2 \# \cdots \# u_m \mid m > n > 0 \wedge u_1, u_2, \ldots, u_m \in a^* b^* c^* \wedge u_n \in B\},$$

in words, the number of a at the beginning indicates where to find a block in B. This language is in OCA and PA but in neither DetOCA nor DetPA. It is, however, in DetPOCA: one would use the counter to find u_n (with n the number of a at the beginning), and check that $u_n \in B$ using the semilinear constraint.

4 Pumping lemmas: Statements and Applications

Although pumping lemmas abound for context-free languages, and thus for OCA, there is no known technique in the PA world that takes any PA language L and a long enough word in L, and creates a word of different length in L. As mentioned in the introduction, the main expressiveness lemma for PA relies on the fact that any cycle taken twice can be moved around without changing membership. Relying on this and by carefully analyzing the behavior of the counter on long enough runs, we can show an expressiveness lemma reminiscent of a pumping property:

▶ **Lemma 4** (Depumping lemma). *Let $L \subseteq \Sigma^*$ be a POCA (resp. DetPOCA) language. For any infinite language $K \subseteq [\varepsilon]_L$ and $N \in \mathbb{N}$, there are words $u, (u_i)_{i \leq n}, v, (v_j)_{j \leq m}$, with $m, n \geq N$, such that all of the following hold:*

1. *$x = (u_1 u)(u_2 u) \cdots (u_n u) \cdot (v_1 v)(v_2 v) \cdots (v_{m-1} v) v_m$ is in K,*
2. *$uv \neq \varepsilon$,*
3. *There exist $w_1, w_2 \in \Sigma^*$ such that, letting $x' = u_1 \cdots u_n v_1 \cdots v_m$, it holds that $w_1 x' w_2$ is in L (resp. in $[\varepsilon]_L$).*

Proof sketch. Let $\mathcal{A}$ be a POCA recognizing L. Now, let $\{x_1, x_2, \ldots\} \subseteq [\varepsilon]_L$ be an infinite set of words such that $\text{length}(x_{i+1}) > \text{length}(x_i) > i$ for all $i \geq 1$. Since $x_i \in [\varepsilon]_L$, for all $i \geq 1$, we know that for any k, there is a run ρ_k of $\mathcal{A}$ on $x_1 \cdots x_k$ which can be extended to an accepting run. Write ρ_k as $\pi_1 \cdots \pi_k$ with each π_i being the subrun corresponding to the subword x_i. (Note that π_i may be different for each value of k.)

For a word w, we say a (sub)run of $\mathcal{A}$ on w that starts and ends in the same state is a *w-cycle*. Since the x_i are increasing in length, so are the π_i. If we take a large enough $k \in \mathbb{N}$, the pigeonhole principle will ensure the existence of a nonempty word u and an index $i \in \mathbb{N}$ such that u-cycles appear more than N times in π_i. Note that it would suffice to argue that we can shift these u-cycles to the other subruns π_j with $j \neq i$ while preserving the validity of the run. (The reason we want to shift cycles rather than just remove them is because we want to preserve the Parikh image.) In this case, w_1 and w_2 would be the words labelling the runs $\pi'_1 \cdots \pi'_{i-1}$ and $\pi'_{i+1} \cdots \pi'_p \sigma$ where: p is the maximal index j such that we shift some u-cycle to π_j, π'_j is the run we get from π_j after shifting the u-cycles, and σ is any run such that $\pi_1 \pi_2 \cdots \pi_p \sigma$ is accepting. The only obstruction in doing this is that, while shifting the u-cycles, we might invalidate the run by making the counter value nonzero at a zero-value transition. This is why we have to take a (possibly) larger $k \in \mathbb{N}$ so that, again by the pigeonhole principle, we are ensured of the existence of another word v such that v-cycles can be shifted along with u-cycles to guarantee that the above does not happen. The technical aspect of the proof lies in formalising this idea. ◀

▶ Remark 5. Lemma 4 is a *de*pumping lemma: it removes portions of the word x that appear often. We forego the statement for the *pumping* lemma as we will not need it, but the same proof shows that we can swap the roles of x and x' in the lemma, thus creating a longer word.

▶ Remark 6. The proof allows for some slightly stronger variations of this statement. For instance, we can assume that K^* is a set of prefixes of words in L instead of assuming that $K \subseteq [\varepsilon]_L$. In this case, the conclusion in the POCA case would not change and, for DetPOCA, the third conclusion would state that $w_1 x' w_2$ is Myhill-Nerode equivalent to *some* word in K.

We now turn to examples of languages that we will show to be outside of DetPOCA and POCA. For the rest of this section, let $\Sigma = \{a, b, c, \#\}$. Our examples will rely on the following languages:

- $L_{ab} = \{\#a^n\#b^n\#c^m\# \mid n, m \in \mathbb{N}\}$,
- $L_{bc} = \{\#a^n\#b^m\#c^m\# \mid n, m \in \mathbb{N}\}$,
- $L_{abc} = \{\#a^n\#b^n\#c^n\# \mid n \in \mathbb{N}\} = L_{ab} \cap L_{bc}$.

▶ **Proposition 7.** *$L_{ab}^* \cup L_{bc}^*$ is not recognizable by a DetPOCA.*

Proof. We rely on Lemma 4. Assume that it is recognizable by a DetPOCA, pick L_{abc} as the language K in Lemma 4, and set $N = 4$. Note that we have indeed that any word of L_{abc} is Myhill-Nerode equivalent to ε in $L_{ab}^* \cup L_{bc}^*$. Using the notations of the lemma, we see that neither u nor v can contain $\#$, since they are both repeated at least 4 times in x. Thus, removing the repetitions of u and v from x, we obtain that x' is of the shape $\#a^i\#b^j\#c^k\#$ but outside of L_{abc}; note that x' has at least one letter from $\{a, b, c\}$. Assume that $i \neq j$ (the case $j \neq k$ is similar), and let $z = \#a\#b\#cc\#$, a word in L. For any words w_1, w_2, the word $w_1 x' w_2 z$ *cannot* be in L, since $x \notin L_{ab}$ and $z \notin L_{bc}$. This shows that $w_1 x' w_2$ is not Myhill-Nerode equivalent to ε, a contradiction. ◀

▶ **Proposition 8.** *Let $B = \{\#a^i\#b^j\#c^k\# \mid i \neq j \vee j \neq k\}$ and $C = (\#a^*\#b^*\#c^*\#)^*$. The language $C \cdot B \cdot C$ is not expressible with a DetPOCA.*

Proof. Note that $L_{abc} \subseteq [\varepsilon]_{CBC}$: indeed, a word in L_{abc} is in $C \setminus B$. We can thus follow the proof of Proposition 7 and stop at the point where x' is seen to be of the shape $\#a^i\#b^j\#c^k\#$ but not in L_{abc}. In fact, $x' \in B$. The conclusion of the pumping lemma now tells us that

there are words w_1, w_2 such that $w_1x'w_2 \in [\varepsilon]_{CBC}$. Since $\varepsilon \cdot x' \in CBC$, we have that $w_1x'w_2x' \in CBC$, so certainly $w_1x'w_2 \in C$ (recall that $B \subseteq C$). But since $x' \in B$, we have that $w_1x'w_2 \in CBC$ too, and thus that $\varepsilon \in CBC$, a contradiction. ◀

▶ Remark 9. The two previous languages happen to be expressible with POCA; let us investigate where the proof of inexpressibility breaks down when we rely on the conclusion in the POCA case of Lemma 4. In Proposition 7, we have the luxury of picking a z: this is not allowed in the POCA case, which simply guarantees that there are words w_1, w_2 such that $w_1x'w_2 \in L$. And indeed, we only know that x' has an unbalanced number of a, b, c, leaving the possibility, that $x' \in L_{ab} \cup L_{bc}$, so $w_1 = w_2 = \varepsilon$ would satisfy the conclusion of the Lemma. Similarly, in Lemma 4, there is no contradiction to be gained from $x' \in B$ on its own, which is the case when $w_1 = w_2 = \varepsilon$ in the Lemma.

▶ **Proposition 10.** *$L^*_{abc} = L^*_{ab} \cap L^*_{bc}$ is not recognizable by a POCA.*

Proof. This follows the same process as the proof of Proposition 7, up to the point where x' is seen to be of the shape $\#a^i\#b^j\#c^k\#$ but not in L_{abc}. Certainly, then, whichever words were to be put at the beginning and end of this word, we cannot reach a word in L. ◀

5 Expressiveness

5.1 Normal forms and inclusion in logarithmic space

We first note that cycles of ε-transitions, in which a POCA would guess a large number, are not needed. Additionally, we can complete the underlying automaton of a POCA so that:

▶ **Theorem 11.** *For any (Det)POCA language L, there is a (Det)POCA A with language L and a constant $c \in \mathbb{N}$ such that any word $w \in L$ is accepted by A with a run of length at most $c|w|$ that reads the whole input. In particular, all cycles of ε-transitions in A decrease the counter.*

We will assume henceforth that all of our POCA are of the above type. From this, and noting that the Parikh constraint of a POCA can be made quantifier-free by adding "equality modulo m" predicates, we immediately get:

▶ **Corollary 12.** *Any DetPOCA language is in the class L. Any POCA language is in the class NL.*

5.2 Separations

We compare the expressiveness of POCA with PA and OCA, in both their deterministic and nondeterministic variants; we obtain the following separations:

▶ **Theorem 13.**
- *DetPA and DetOCA are strictly less expressive than DetPOCA;*
- *PA and OCA are strictly less expressive than POCA;*
- *The expressive power of DetPOCA is incomparable with that of PA and OCA.*

Proof. The inclusions in the first two statements are immediate and strictness was mentioned in Example 1. This example also shows that DetPOCA is not included in either PA or OCA.

What is left to show is that there are PA and OCA languages that are not in DetPOCA. The language CBC of Proposition 8 is such a language: it is not in DetPOCA, but it can be expressed with both a PA and an OCA (one simply guesses when the B word occurs and, within this word, guesses and checks which of the options $i \neq j$ or $j \neq k$ holds). ◀

In addition, we have exhibited (Propositions 7 and 8) languages expressible with POCA but not DetPOCA. Hence:

▶ **Theorem 14.** *DetPOCA are strictly less expressive than POCA.*

5.3 Rendering the OCA or Parikh part useless in a POCA

The wide disparity of expressive power between POCA and its base models OCA and Parikh automata stands in sharp contrast with the intuition that the two counting features of POCA are orthogonal. In this section, we explore how we can essentially "saturate" the abilities of one of the base models, in such a way that it cannot contribute meaningfully to recognizing a language. This is reminiscent of the work of [18], in which it is shown that if the shuffle of a nonregular context-free language and another language T is still context-free, then T has to be regular. In other words, the nonregular context-free language "saturates" the stack. See also [15] for a related notion of *simplest nonregular context-free language*.

Recall that the *shuffle* of two words $u \sqcup\!\sqcup v$ is the set $\{u_1v_1 \cdots u_kv_k \mid u_i, v_i \in \Sigma^* \wedge u = u_1 \cdots u_k \wedge v = v_1 \cdots v_k\}$. The shuffle of two languages is the set of shuffles of words from each language.

▶ **Theorem 15** (One-counter-stripping)**.** *Let $L' \subseteq \Sigma^*$ and $a, b \notin \Sigma$. If $L' \sqcup\!\sqcup \{a^nb^n \mid n \geq 0\}^*$ is a (Det)POCA language, then L' is a (Det)PA language. The converse holds.*

Proof sketch. Let the POCA accepting L be $\mathcal{A}$. To prove the theorem above, we give a procedure of producing the intended Parikh automaton $\mathcal{A}'$ accepting L', using $\mathcal{A}$ as a black box. The main idea is that $\mathcal{A}'$ simulates $\mathcal{A}$, but along the accepting runs in $\mathcal{A}'$, the counter is only used in a bounded way that can be encoded in the state space; hence we can get rid of the counter while accepting L'. We formalize this idea as follows:

Let C, C_ε be the number of simple cycles with nonempty underlying word, and simple ε-cycles in $\mathcal{A}$. Let $C' = C + C_\varepsilon + 1$. Define a function $\mathrm{pad} : \Sigma^* \to (\Sigma \uplus \{a, b\})^*$ such that for any word $w = c_1 \cdots c_n \in \Sigma^*$, $\mathrm{pad}(w)$ is of the form $\mathrm{pad}(w) = c_1 \cdot \left(a^{|Q|}b^{|Q|}\right)^{C'} \cdots c_n \cdot \left(a^{|Q|}b^{|Q|}\right)^{C'}$. Note that, if $w \in L'$, then $\mathrm{pad}(w) \in L$. We extend the function on the language L' naturally: $\mathrm{pad}(L) = \{\mathrm{pad}(x) \mid x \in L\} \subset L'$. The following lemma then holds.

▶ **Lemma 16.** *Let $Reach = \{n \mid \exists (x \cdot y) \in \mathrm{pad}(L')$, and $q_f \in F, (q_0, 0) \xrightarrow{x} (q, n) \xrightarrow{y} q_f\}$ be the set of all counter values appearing in any accepting run in $\mathcal{A}$ on reading a word from $\mathrm{pad}(L')$. The set Reach is bounded, i.e., there exists a bound $B \in \mathbb{N}$ such that, for every $n \in Reach, n \leq B$.*

Using the above lemma, we can already outline the procedure to construct the desired PA $\mathcal{A}'$ such that $L(\mathcal{A}') = L'$. Let the POCA $\mathcal{A}$ be of the form $(Q, q_0, \Sigma \uplus \{a, b\}, \Delta, F, \varphi)$. Then the PA $\mathcal{A}'$ is of the form $(Q', q_0', \Sigma, \Delta', F', \varphi')$ such that,

- $Q' = Q \times \{0, 1, \ldots, B\}) \uplus \{r\}$,
- $q_0' = (q_0, 0)$ and $F' = F \times \{0, 1, \ldots, B\}$,
- for every run of the form $(q, i) \xrightarrow{l \cdot \left(a^{|Q|}b^{|Q|}\right)^{C'}} (q', j) \in \Delta^*$, where $l \in \Sigma$, $q, q' \in Q$, and $i \leq B$

(i). if $j \leq B$, then $\left((q, i) \xrightarrow{l} (q', j)\right) \in \Delta'$

(ii). otherwise $\left((q, i) \xrightarrow{l} r\right) \in \Delta'$.

Note that, for every configuration (q, i) in $\mathcal{A}$ and $l \in \Sigma$, the number of runs on the word $l \cdot \left(a^{|Q|}b^{|Q|}\right)^{C'}$ is finite as $\mathcal{A}$ does not contain any nonnegative ε-cycle.

- The construction of Δ' from Δ imposes a linear function $f : \mathbb{Z}^{\Delta'} \to \mathbb{Z}^{\Delta}$. Let the Parikh constraint for $\mathcal{A}$ be φ with $|\Delta|$ free variables. Then we define φ' with $|\Delta'|$ free variables such that, a vector $\boldsymbol{v} \in \mathbb{Z}^{\Delta'} \models \varphi'$ if and only if $f(\boldsymbol{v}) \models \varphi$.

It is easy to check that $\mathcal{A}'$ accept w if and only if $\mathcal{A}$ accepts $\mathrm{pad}(w)$. This implies $\mathcal{A}'$ recognises L'. The above procedure also preserves determinacy, i.e., if $\mathcal{A}$ is deterministic, then $\mathcal{A}'$ is also deterministic. ◀

Similarly, we expect that some operation would render the Parikh constraint useless. Specifically, assume that for some language L, the language $(L\#)^*$ is recognized by a POCA A. Consider a long word w in $L\#$; in the accepting run for it, we could move (pairs of) cycles *after* the # symbol. Repeating this, we obtain a subword of w of constant length, appearing before #, which must also be in L. Hence to recognize L, one only needs to simulate A for a counter-correct run and find the subword of constant length to check for constraint-correctness. The Parikh constraint can thus be hardcoded, and we thus conjecture:

▶ **Conjecture 17.** *Let $L \subseteq \Sigma^*$ be a language and $\# \notin L$. If $(L\#)^*$ is a POCA (resp. DetPOCA) language, then L is an OCA (resp. DetOCA) language.*

6 Closure Properties

We study in this section the closure properties of the POCA classes. We start with positive closure properties and then move to nonclosure claims. The results of this section are summarized in Figure 1, in the Introduction.

6.1 Positive closure properties

▶ **Theorem 18.** *The class of languages recognized by DetPOCA is closed under complement, inverse morphisms, and intersection/union with regular languages.*

Proof.

Complement. When there is no counter, i.e., in the DetPA case, this is fairly straightforward: the complement of a DetPA is the union of

1. the complement of the language of the underlying automaton; and
2. the language of the DetPA with the semilinear constraint negated.

For DetPOCA, this approach has multiple caveats: as we will see, DetPOCA is not closed under union, moreover, we need to take the complement of the underlying *DetOCA* which is slightly more technical. Let us sketch how to overcome these limitations. Consider a DetPOCA $\mathcal{A}$. We follow the standard first steps for complementing deterministic pushdown automata (e.g., [13, Chapter 10.2]):

1. Ensure that the automaton reads the input word in its entirety;
2. Mark as final any state that can reach a final state following only ε-transitions (possibly zeroing the counter along the way).

The language of $\mathcal{A}$ is unchanged and a word is rejected iff after reading it, we reach a nonfinal state. Call F the set of final states at this point. Next, note that given the Parikh image of a run starting in the initial state, one can find, in first-order logic, which state is the last one in the run (this is the one state that has more incoming transitions taken than outgoing ones). Let us define a DetPOCA $\mathcal{B}$ as $\mathcal{A}$, but with *all* states final. In addition, the constraint formula of $\mathcal{B}$ accepts if either the last state of the run is *not* in F or if it is and the constraint

formula of $\mathcal{A}$ *rejected* the Parikh image. A word is accepted by $\mathcal{B}$ if either the underlying DetOCA of $\mathcal{A}$ rejected it or if it did accept it but the constraint set of $\mathcal{A}$ was rejecting it. This is precisely the complement of $\mathcal{L}(\mathcal{A})$, concluding this proof.

Intersection/union with regular languages, inverse morphism. These are standard. ◀

▶ **Theorem 19.** *The class of languages recognized by POCA is closed under union, concatenation, morphisms, inverse morphisms, and intersection with regular languages.*

Proof.

Concatenation. This follows the classical construction for regular languages. The only complication is that the counter has to be reset after the jump (and the Parikh constraint has to undergo variable renaming as they speak about disjoint sets of variables arising from the original automata). This can be achieved using ε-transitions with decrements on the counter.

Other closures. These follow standard arguments. ◀

6.2 Nonclosure properties

▶ **Theorem 20.** *The class of languages recognized by DetPOCA is not closed under union, intersection, concatenation with regular languages, and morphisms.*

Proof.

Union, intersection, concatenation with regular languages. This is covered by Propositions 7, 8, and 10, respectively, noting that L_{ab}^*, L_{bc}^*, B, and C (using the notation therein) are all expressible using DetPOCA.

Morphisms. This is immediate from nonclosure under union. Indeed, take any two DetPOCA languages L_1, L_2 over the same alphabet Σ and let Γ be a disjoint alphabet of the same size as Σ. Let $h\colon \Gamma^* \to \Sigma^*$ be any bijective morphism. Certainly, $T = L_1 \cup h^{-1}(L_2)$ is recognizable with a DetPOCA, since one can decide which language to test for by reading the first letter. Extending h so that it is the identity over Σ, we have, however, that $h(T) = L_1 \cup L_2$. Thus if DetPOCA were closed under morphism, it would be closed under union, a contradiction. ◀

▶ **Theorem 21.** *The class of languages recognized by POCA is not closed under intersection and complement.*

Proof. For intersection, this is the contents of Proposition 10. Nonclosure under complement is immediate from closure under union but not under intersection. ◀

7 Decision Problems

7.1 Classical decision problems

In this section, we study the computational complexity of some classical decision problems for automata: given one or two automata $\mathcal{A}$ and $\mathcal{B}$, *nonemptiness* asks whether $\mathcal{L}(\mathcal{A}) \neq \emptyset$, *universality* whether $\mathcal{L}(\mathcal{A}) = \Sigma^*$, *inclusion* whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$, and *equivalence* whether $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. From the known results listed in Figure 1, we immediately get:

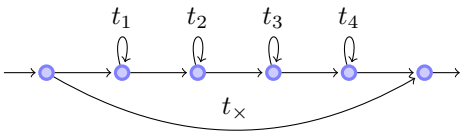
▶ **Corollary 22.** *Universality, inclusion, and equivalence are undecidable for POCA. Inclusion is undecidable for DetPOCA.*

▶ **Theorem 23.** *Nonemptiness is NP-complete for DetPOCA and POCA. Universality is coNP-complete for DetPOCA.*

Proof. For nonemptiness, hardness comes from the same property for DetPA and PA. In [7, Proposition III.2], this is shown by encoding instances of the SubsetSum problem into an instance of the nonemptiness problem for DetPA. NP-membership for nonemptiness for POCA can be shown in a way similar to [7, Proposition III.2]: we can construct an existential Presburger formula φ_ρ whose models form the set of Parikh images of accepting runs of the underlying OCA of a given POCA. The formula φ_ρ can be obtained in polynomial time using a construction from [22, Theorem 4].[2] We can then check whether $\varphi \wedge \varphi_\rho$ is satisfiable, which is the case if and only if the language of the given POCA is nonempty. Since satisfiability of existential Presburger formulas is in NP (see, e.g., [10] and references therein), this concludes the proof.

For universality of DetPOCA, this is a direct consequence of the (effective!) closure of DetPOCA under complement (Theorem 18) and the previous discussion. ◀

▶ Remark 24. NP-hardness of nonemptiness has only little to do with the hardness of solving the constraint formula itself or from encoding numbers in that formula in binary. Indeed, the constraint formula obtained in the reduction from SubsetSum can be made quantifier-free and without constants besides 1, in which case checking that a tuple satisfies the formula is easy (in L). Let us make this reduction more explicit to see this. We build a partial PA (the counter is not needed) that either "takes" a number from the instance set or does not, in the sense that for each number n in the set, there will be a transition t that is either taken n times or 0 times. We present the construction through an example: The following partial PA will select whether $n = 13$ gets into the candidate subset:



The constraint formula would assert (writing t_i for the number of times t_i is taken):

$$t_\times = 1 \vee ((t_1 = 1) \wedge (t_2 = t_1 + t_1 + 1) \wedge (t_3 = t_2 + t_2) \wedge (t_4 = t_3 + t_3 + 1)),$$

corresponding to the binary encoding of 13, that is, 1101. Transition t_4 is thus taken exactly 13 times or not at all, and the reduction is concluded by summing these selection transitions and checking if they are unequal to the target value.

▶ **Corollary 25.** *It is coNP-complete to decide, given a POCA $\mathcal{A}$ and a regular language R as an NFA, whether $\mathcal{L}(\mathcal{A}) \subseteq R$.*

▶ **Open Question 26.** *Is equivalence decidable for DetPOCA? We conjecture that it is.*

7.2 Parametric decision problems

In the field of formal verification, computational models represent so-called *reactive systems* that communicate with and evolve based on their surrounding environment. To formalize the varying conditions provided by the environment, the models receive *parameters*, usually

[2] See [12] for a construction that fixes a small mistake in the proof of that theorem.

integer valued variables [1]. In [3], it is shown that reachability in parametric timed automata with two clocks correspond to parametric emptiness problem for certain classes of one-counter machines. In [8] and [19], the authors showed that parametric emptiness for OCA ("for any parameter value, the OCA is nonempty") is decidable, where the parameters may appear on the updates of the counter. We study that problem for POCA; there are two natural places where parameters could appear: the counter updates and within the constraint formula.

▶ **Definition 27.** *A* parametric POCA *is a tuple* $\mathcal{A} = (Q, q_0, \Sigma, X, \Delta_0, \Delta_+, F, \varphi)$ *where:*

- Q, q_0, Σ, *and* F *are as in POCA,*
- $X = \{x_1, \ldots, x_n\}$ *is a finite set of* parameters *that will take integer values,*
- Δ_0 *and* Δ_+ *are as in POCA but can also include tuples of the form* $(p, a, +x, q)$ *and* $(p, a, -x, q)$ *with* $a \in \Sigma^\varepsilon$ *and* $x \in X$*,*
- φ *has* $d + n$ *free variables, where* $d = |\Delta_0| + |\Delta_+|$*, corresponding to the Parikh image of the run* and *the valuation of the parameters in* X*.*

Given a valuation $\mu\colon X \to \mathbb{N}$*,* $\mathcal{A}$ *induces a POCA* $\mathcal{A}_\mu$ *with well-defined runs, language, etc. A* POCA with parametric updates *is a parametric POCA in which* φ *does not have occurrence of the parameters and a* POCA with parametric constraint *is a parametric POCA in which* Δ_0 *and* Δ_+ *only have nonparametric transitions.*

Given a parametric POCA $\mathcal{A}$ with parameter set X, the *parametric universal nonemptiness problem*, PUNE for short, asks whether it holds that, for all $\mu\colon X \to \mathbb{N}$, we have $\mathcal{L}(\mathcal{A}_\mu) \neq \emptyset$.

▶ **Theorem 28.** *The* PUNE *problem for POCA with parametric updates is undecidable. It is decidable and complete for coNEXP*^{EXP} *for POCA with parametric constraint.*

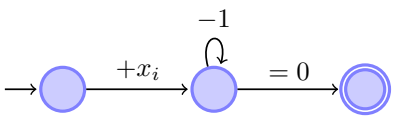
Proof.

Parametric updates. We present a reduction from Hilbert's tenth Problem to the PUNE problem. Recall that Hilbert's Tenth Problem asks, given a polynomial with integer coefficients, if it has a positive integer solution.

Let $P(x_1, \ldots, x_n)$ be such a polynomial and write $P = c_1 M_1 + \cdots + c_k M_k$ with each c_i in $\mathbb{Z}$ and each M_i a monomial with coefficient 1 (e.g., $x_1 x_2^2$). We construct a POCA $\mathcal{A}$ with parametric updates over the parameter set $\{x_1, \ldots, x_n\}$ that *evaluates* P. This is in the following sense: there are transitions $t_1, \ldots, t_k$ of $\mathcal{A}$ such that for any valuation μ of the parameters, there is a unique accepting run ρ in $\mathcal{A}_\mu$, and, writing $|\rho|_{t_i}$ for the number of times t_i occurs in ρ:

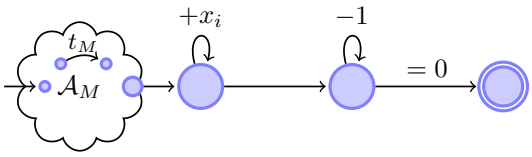
$$c_1|\rho|_{t_1} + \cdots + c_k|\rho|_{t_k} = P\left(\mu(x_1), \ldots, \mu(x_n)\right).$$

We start with the simplest case: $P = x_i$. Consider the following OCA (the *labels* are not important, so we assume that each transition has a unique label and do not write it):



Here, our transition t that evaluates to $P(\mu(x_i))$ is simply the self-loop: if the run is counter-correct, this loop must have been taken $\mu(x_i)$ times. Note that accepting runs end with a counter value of zero and that this POCA has a single final state – these are properties we will keep throughout this construction.

Next, assume $P = Mx_i$ with M a monomial with coefficient 1. We assume that we have built a POCA $\mathcal{A}_M$ with a transition t_M that is taken $M(x_1, \ldots, x_n)$ times on accepting runs. We then build the following POCA for P:



As constraint, we combine the Parikh constraint of $\mathcal{A}_M$ with the statement that the $+x_i$ loop should be taken the same number of times as t_M; consequently, for any accepting run ρ of this POCA with valuation μ, the -1 loop is taken $|\rho|_{t_M}\mu(x_i)$ times, which is $M(\mu(x_1),\dots,\mu(x_n))\mu(x_i)$ by hypothesis. This -1 loop is thus the transition that evaluates to $P(\mu(x_1),\dots,\mu(x_n))$.

For the general case, we can chain together our POCA for each monomial one after the other, and obtain our claimed POCA for any polynomial. The constraint formula can then compute the exact value of $P(\mu(x_1),\dots,\mu(x_n))$ and accept if it is nonzero. Thus, there is no positive integer solution to P iff for any valuation μ, $\mathcal{A}_\mu$ has a nonempty language.

Parametric constraint. This follows the same proof as decidability in the nonparametric case, but results in a formula of Presburger arithmetic with one alternation starting with $\forall$. Validity of such sentences is complete for the class mentioned in the statement of the theorem [9]. ◀

8 Conclusion

In the long tradition of combining computational means to obtain expressive models (e.g., [20]), we have equipped one-counter automata with a mechanism to count events *globally*. This mechanism, namely constraining the Parikh image of runs to fall within a semilinear set, always enables recognizing non-context-free languages (e.g., $\{a^nb^nc^n \mid n > 0\}$) while still preserving the decidability of emptiness for models with effective semilinear Parikh images. However, studying the expressiveness of the combined model is surprisingly difficult: techniques that apply to the original model usually do not preserve satisfaction of the semilinear constraint.

Here, we have obtained expressiveness lemmas that allowed us to study the closure properties and the class relationships of the models at play. In particular, we have shown that there are languages expressible with a combination of deterministic one-counter automata and semilinear constraint that cannot be obtained by any of the underlying mechanisms: the whole is greater than the sum of its parts.

We underline research directions stemming from this work:

- We left open two main questions: 1. Is equivalence decidable for DetPOCA? We conjecture that a refinement of the algorithms for DetOCA ([21, 2]) will lead to a positive answer. 2. Is it true that if $L' = (L\#)^*$ is a POCA language, then L' doesn't use the semilinear constraint in any meaningful way, so that L is an OCA language?
- The undecidability of the parametric universality nonemptiness problem (Theorem 28) is rather unfortunate. Indeed, we had originally expected that this problem would be a natural automata counterpart of the validity of sentences in a fragment of Presburger arithmetic with divisibility called BIL (see [19, Conclusion]). Such sentences are naturally translated to POCA with parametric updates in such a way that validity corresponds to parametric nonemptiness, but alas, validity of the BIL fragment is decidable while the Pune problem for POCA with parametric updates is not. For context, an elegant connection between another fragment of Presburger arithmetic with divisibility and OCA with parametric updates was established in [11]: the validity problem of the former is interreducible with the nonemptiness problem of the latter via nondeterministic polynomial-time reductions. This leaves open the problem of finding such a correspondence for BIL.

- A reviewer asked the following relevant question: Can a POCA for an OCA language be more succinct than the equivalent OCA (and similarly for a POCA for a PA language)? Some examples come to mind: $\{w \in \{a,b\}^* \mid |w|_a = 18|w|_b\}$ requires an OCA with at least 18 states, but can be done with a PA with one state. A more interesting class of languages to study this trade-off is that of languages expressible with an OCA but not by a PA, or conversely.

References

1. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 592–601. ACM, 1993. doi:10.1145/167088.167242.
2. Stanislav Böhm, Stefan Göller, and Petr Jancar. Equivalence of deterministic one-counter automata is NL-complete. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 131–140. ACM, 2013. doi:10.1145/2488608.2488626.
3. Daniel Bundala and Joël Ouaknine. On parametric timed automata and one-counter machines. *Inf. Comput.*, 253:272–303, 2017. doi:10.1016/j.ic.2016.07.011.
4. Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Affine Parikh automata. *RAIRO – Theoretical Informatics and Applications*, 46(04):511–545, 2012. doi:10.1051/ita/2012013.
5. Michaël Cadilhac, Andreas Krebs, and Pierre McKenzie. The algebraic theory of parikh automata. *Theory of Computing Systems*, 62(5):1241–1268, 2017. doi:10.1007/s00224-017-9817-2.
6. Ehsan Chiniforooshan, Mark Daley, Oscar H. Ibarra, Lila Kari, and Shinnosuke Seki. One-reversal counter machines and multihead automata: Revisited. In *Current Trends in Theory and Practice of Computer Science*, volume 6543 of *LNCS*, pages 166–177. Springer, 2011.
7. Diego Figueira and Leonid Libkin. Path logics for querying graphs: Combining expressiveness and efficiency. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 329–340. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.39.
8. Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Model checking succinct and parametric one-counter automata. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2010. doi:10.1007/978-3-642-14162-1_48.
9. Christoph Haase. Subclasses of presburger arithmetic and the weak EXP hierarchy. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 – 18, 2014*, pages 47:1–47:10. ACM, 2014. doi:10.1145/2603088.2603092.
10. Christoph Haase. A survival guide to Presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. doi:10.1145/3242953.3242964.
11. Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009 – Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, volume 5710 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2009. doi:10.1007/978-3-642-04081-8_25.
12. Matthew Hague and Anthony Widjaja Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification – 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2012. doi:10.1007/978-3-642-31424-7_22.

13 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

14 Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978. doi:10.1145/322047.322058.

15 Petr Jancar and Jiří Šíma. The simplest non-regular deterministic context-free language. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 63:1–63:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.63.

16 Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In *International Colloquium on Automata, Languages and Programming*, volume 2719 of *LNCS*, pages 681–696. Springer-Verlag, 2003. doi:10.1007/3-540-45061-0_54.

17 Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for *AC*-like equational theories with homomorphisms. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 2005. doi:10.1007/978-3-540-32033-3_23.

18 Michel Latteux and Grzegorz Rozenberg. Commutative one-counter languages are regular. *J. Comput. Syst. Sci.*, 29(1):54–57, 1984. doi:10.1016/0022-0000(84)90013-8.

19 Guillermo A. Pérez and Ritam Raha. Revisiting parameter synthesis for one-counter automata. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 33:1–33:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CSL.2022.33.

20 Yann-Joachim Ringard. Mustard watches: An integrated approach to time and food. Technical report, Équipe de Logique Mathématique, Paris 7, 1990. URL: https://girard.perso.math.cnrs.fr/mustard/article.html.

21 Leslie G. Valiant and Michael S. Paterson. Deterministic one-counter automata. *Journal of Computer and System Sciences*, 10(3):340–350, 1975. doi:10.1016/S0022-0000(75)80005-5.

22 Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational horn clauses. In Robert Nieuwenhuis, editor, *Automated Deduction – CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2005. doi:10.1007/11532231_25.