Parikh Automata over Infinite Words

Guha, Shibashis; Jecker, Ismaël; Lehtinen, Karoliina; Zimmermann, Martin

Published in:
42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022

DOI (link to publication from Publisher):
10.4230/LIPIcs.FSTTCS.2022.40

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

Parikh Automata over Infinite Words

Shibashis Guha
Tata Institute of Fundamental Research, Mumbai, India

Ismaël Jecker
University of Warsaw, Poland

Karoliina Lehtinen
CNRS, Aix-Marseille University, LIS, Marseille, France

Martin Zimmermann
Aalborg University, Denmark

Abstract

Parikh automata extend finite automata by counters that can be tested for membership in a semilinear set, but only at the end of a run, thereby preserving many of the desirable algorithmic properties of finite automata. Here, we study the extension of the classical framework onto infinite inputs: We introduce reachability, safety, Büchi, and co-Büchi Parikh automata on infinite words and study expressiveness, closure properties, and the complexity of verification problems.

We show that almost all classes of automata have pairwise incomparable expressiveness, both in the deterministic and the nondeterministic case; a result that sharply contrasts with the well-known hierarchy in the ω-regular setting. Furthermore, emptiness is shown decidable for Parikh automata with reachability or Büchi acceptance, but undecidable for safety and co-Büchi acceptance. Most importantly, we show decidability of model checking with specifications given by deterministic Parikh automata with safety or co-Büchi acceptance, but also undecidability for all other types of automata. Finally, solving games is undecidable for all types.

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

Keywords and phrases Parikh automata, ω-automata, Infinite Games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.40

Related Version *Full Version*: `https://arxiv.org/abs/2207.07694` [17]

Funding *Shibashis Guha*: Supported by the DST-SERB project SRG/2021/000466.
Ismaël Jecker: Supported by the ERC grant 950398 (INFSYS).
Martin Zimmermann: Supported by DIREC – Digital Research Centre Denmark.

Acknowledgements We want to thank an anonymous reviewer for proposing Lemma 14.

1 Introduction

While finite-state automata are the keystone of automata-theoretic verification, they are not expressive enough to deal with the many nonregular aspects of realistic verification problems. Various extensions of finite automata have emerged over the years, to allow for the specification of context-free properties and beyond, as well as the modelling of timed and quantitative aspects of systems. Among these extensions, Parikh automata, introduced by Klaedtke and Rueß [19], consist of finite automata augmented with counters that can only be incremented. A Parikh automaton only accepts a word if the final counter-configuration is within a semilinear set specified by the automaton. As the counters do not interfere with the control flow of the automaton, that is, counter values do not affect whether transitions are enabled, they allow for mild quantitative computations without the full power of vector addition systems or other more powerful models.

For example, the nonregular language of words that have more a's than b's is accepted by a Parikh automaton obtained from the one-state DFA accepting $\{a, b\}^*$ by equipping it with two counters, one counting the a's in the input, the other counting the b's, and a semilinear set ensuring that the first counter is larger than the second one. With a similar approach, one can construct a Parikh automaton accepting the non-context-free language of words that have more a's than b's and more a's than c's.

Klaedtke and Rueß [19] showed Parikh automata to be expressively equivalent to a quantitative version of existential WMSO that allows for reasoning about set cardinalities. Their expressiveness also coincides with that of reversal-bounded counter machines [19], in which counters can go from decrementing to incrementing only a bounded number of times, but in which counters affect control flow [18]. The (weakly) unambiguous restriction of Parikh automata, that is, those that have at most one accepting run, on the other hand, coincide with unambiguous reversal-bounded counter machines [2]. Parikh automata are also expressively equivalent to weighted finite automata over the groups $(\mathbb{Z}^k, +, 0)$ [9, 21] for $k \geqslant 1$. This shows that Parikh automata accept a natural class of quantitative specifications.

Despite their expressiveness, Parikh automata retain some decidability: nonemptiness, in particular, is NP-complete [12]. For weakly unambiguous Parikh automata, inclusion [5] and regular separability [6] are decidable as well. Figueira and Libkin [12] also argued that this model is well-suited for querying graph databases, while mitigating some of the complexity issues related with more expressive query languages. Further, they have been used in the model checking of transducer properties [14].

As Parikh automata have been established as a robust and useful model, many variants thereof exist: pushdown (visibly [8] and otherwise [23]), two-way with [8] and without stack [13], unambiguous [4], and weakly unambiguous [2] Parikh automata, to name a few. Despite this attention, so far, some more elementary questions have remained unanswered. For instance, despite Klaedtke and Rueß's suggestion in [19] that the model could be extended to infinite words, we are not aware of previous work on ω-Parikh automata.

Yet, specifications over infinite words are a crucial part of the modern verification landscape. Indeed, programs, especially safety-critical ones, are often expected to run continuously, possibly in interaction with an environment. Then, executions are better described by infinite words, and accordingly, automata over infinite, rather than finite, words are appropriate for capturing specifications.

This is the starting point of our contribution: we extend Parikh automata to infinite inputs, and consider reachability, safety, Büchi, and co-Büchi acceptance conditions. We observe that when it comes to reachability and Büchi, there are two possible definitions: an asynchronous one that just requires both an accepting state and the semilinear set to be reached (once or infinitely often) by the run, but not necessarily at the same time, and a synchronous one that requires both to be reached (once or infinitely often) simultaneously. Parikh automata on infinite words accept, for example, the languages of infinite words

- with some prefix having more a's than b's (reachability acceptance),
- with all nonempty prefixes having more a's than b's (safety acceptance),
- with infinitely many prefixes having more a's than b's (Büchi acceptance), and
- with almost all prefixes having more a's than b's (co-Büchi acceptance).

We establish that, both for reachability and Büchi acceptance, both the synchronous and the asynchronous variant are linearly equivalent in the presence of nondeterminism, but not for deterministic automata. Hence, by considering all acceptance conditions and (non)determinism, we end up with twelve different classes of automata. We show that almost all of these classes have pairwise incomparable expressiveness, which is in sharp contrast

to the well-known hierarchies in the ω-regular case. Furthermore, we establish an almost complete picture of the Boolean closure properties of these twelve classes of automata. Most notably, they lack closure under negation, even for nondeterministic Büchi Parikh automata. Again, this result should be contrasted with the ω-regular case, where nondeterministic Büchi automata are closed under negation [22].

We then study the complexity of the most important verification problems, e.g., nonemptiness, universality, model checking, and solving games. We show that nonemptiness is undecidable for deterministic safety and co-Büchi Parikh automata. However, perhaps surprisingly, we also show that nonemptiness is decidable, in fact NP-complete, for reachability and Büchi Parikh automata, both for the synchronous and the asynchronous versions. Strikingly, for Parikh automata, the Büchi acceptance condition is algorithmically simpler than the safety one (recall that their expressiveness is pairwise incomparable).

Next, we consider model checking, arguably the most successful application of automata theory in the field of automated verification. Model checking asks whether a given finite-state system satisfies a given specification. Here, we consider quantitative specifications given by Parikh automata. Model checking is decidable for specifications given by deterministic Parikh automata with safety or co-Büchi acceptance. On the other hand, the problem is undecidable for all other classes of automata.

The positive results imply that one can model-check an arbiter serving requests from two clients against specifications like "the accumulated waiting time between requests and responses of client 1 is always at most twice the accumulated waiting time for client 2 and vice versa" and "the difference between the number of responses for client 1 and the number of responses for client 2 is from some point onward bounded by 100". Note that both properties are not ω-regular.

Finally, we consider solving games with winning conditions expressed by Parikh automata. Zero-sum two-player games are a key formalism used to model the interaction of programs with an uncontrollable environment. In particular, they are at the heart of solving synthesis problems in which, rather than verifying the correctness of an existing program, we are interested in generating a program that is correct by construction, from its specifications. In these games, the specification corresponds to the winning condition: one player tries to build a word (i.e., behaviour) that is in the specification, while the other tries to prevent this. As with model checking, using Parikh automata to capture the specification would enable these well-understood game-based techniques to be extended to mildly quantitative specifications. However, we show that games with winning conditions specified by Parikh automata are undecidable for all acceptance conditions we consider.

All proofs omitted due to space restrictions can be found in the full version [17].

2 Definitions

An alphabet is a finite nonempty set Σ of letters. As usual, ε denotes the empty word, Σ^* (Σ^+, Σ^ω) denotes the set of finite (finite nonempty, infinite) words over Σ. The length of a finite word w is denoted by $|w|$ and, for notational convenience, we define $|w| = \infty$ for infinite words w.

The number of occurrences of the letter a in a finite word w is denoted by $|w|_a$. Let $a, b \in \Sigma$. A word $w \in \Sigma^*$ is (a, b)-balanced if $|w|_a = |w|_b$, otherwise it is (a, b)-unbalanced. Note that the empty word is (a, b)-balanced.

Semilinear Sets. Let $\mathbb{N}$ denote the set of nonnegative integers. Let $\vec{v} = (v_0, \ldots, v_{d-1}) \in \mathbb{N}^d$ and $\vec{v}\,' = (v'_0, \ldots, v'_{d'-1}) \in \mathbb{N}^{d'}$ be a pair of vectors. We define their concatenation as $\vec{v} \cdot \vec{v}\,' = (v_0, \ldots, v_{d-1}, v'_0, \ldots, v'_{d'-1}) \in \mathbb{N}^{d+d'}$. We lift the concatenation of vectors to sets $D \subseteq \mathbb{N}^d$ and $D' \subseteq \mathbb{N}^{d'}$ via $D \cdot D' = \{\vec{v} \cdot \vec{v}\,' \mid \vec{v} \in D \text{ and } \vec{v}\,' \in D'\}$.

Let $d \geqslant 1$. A set $C \subseteq \mathbb{N}^d$ is *linear* if there are vectors $\vec{v}_0, \ldots, \vec{v}_k \in \mathbb{N}^d$ such that

$$C = \left\{ \vec{v}_0 + \sum\nolimits_{i=1}^{k} c_i \vec{v}_i \;\middle|\; c_i \in \mathbb{N} \text{ for } i = 1, \ldots, k \right\}.$$

Furthermore, a subset of $\mathbb{N}^d$ is *semilinear* if it is a finite union of linear sets.

▶ **Proposition 1** ([16]). *If $C, C' \subseteq \mathbb{N}^d$ are semilinear, then so are $C \cup C'$, $C \cap C'$, $\mathbb{N}^d \setminus C$, as well as $\mathbb{N}^{d'} \cdot C$ and $C \cdot \mathbb{N}^{d'}$ for every $d' \geqslant 1$.*

Finite Automata. A (nondeterministic) finite automaton (NFA) $\mathcal{A} = (Q, \Sigma, q_I, \Delta, F)$ over Σ consists of a finite set Q of states containing the initial state q_I, an alphabet Σ, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and a set $F \subseteq Q$ of accepting states. The NFA is deterministic (i.e., a DFA) if for every state $q \in Q$ and every letter $a \in \Sigma$, there is at most one $q' \in Q$ such that (q, a, q') is a transition of $\mathcal{A}$. A run of $\mathcal{A}$ is a (possibly empty) sequence $(q_0, w_0, q_1)(q_1, w_1, q_2) \cdots (q_{n-1}, w_{n-1}, q_n)$ of transitions with $q_0 = q_I$. It processes the word $w_0 w_1 \cdots w_{n-1} \in \Sigma^*$. The run is *accepting* if it is either empty and the initial state is accepting or if it is nonempty and q_n is accepting. The language $L(\mathcal{A})$ of $\mathcal{A}$ contains all finite words $w \in \Sigma^*$ such that $\mathcal{A}$ has an accepting run processing w.

Parikh Automata. Let Σ be an alphabet, $d \geqslant 1$, and D a finite subset of $\mathbb{N}^d$. Furthermore, let $w = (a_0, \vec{v}_0) \cdots (a_{n-1}, \vec{v}_{n-1})$ be a word over $\Sigma \times D$. The Σ-projection of w is $p_\Sigma(w) = a_0 \cdots a_{n-1} \in \Sigma^*$ and its *extended Parikh image* is $\Phi_e(w) = \sum_{j=0}^{n-1} \vec{v}_j \in \mathbb{N}^d$ with the convention $\Phi_e(\varepsilon) = \vec{0}$, where $\vec{0}$ is the d-dimensional zero vector.

A *Parikh automaton* (PA) is a pair $(\mathcal{A}, C)$ such that $\mathcal{A}$ is an NFA over $\Sigma \times D$ for some input alphabet Σ and some finite $D \subseteq \mathbb{N}^d$ for some $d \geqslant 1$, and $C \subseteq \mathbb{N}^d$ is semilinear. The language of $(\mathcal{A}, C)$ consists of the Σ-projections of words $w \in L(\mathcal{A})$ whose extended Parikh image is in C, i.e.,

$$L(\mathcal{A}, C) = \{p_\Sigma(w) \mid w \in L(\mathcal{A}) \text{ with } \Phi_e(w) \in C\}.$$

The automaton $(\mathcal{A}, C)$ is deterministic, if for every state q of $\mathcal{A}$ and every $a \in \Sigma$, there is at most one pair $(q', \vec{v}) \in Q \times D$ such that $(q, (a, \vec{v}), q')$ is a transition of $\mathcal{A}$. Note that this definition does *not* coincide with $\mathcal{A}$ being deterministic: As mentioned above, $\mathcal{A}$ accepts words over $\Sigma \times D$ while $(\mathcal{A}, C)$ accepts words over Σ. Therefore, determinism is defined with respect to Σ only.

Note that the above definition of $L(\mathcal{A}, C)$ coincides with the following alternative definition via accepting runs: A run ρ of $(\mathcal{A}, C)$ is a run

$$\rho = (q_0, (a_0, \vec{v}_0), q_1)(q_1, (a_1, \vec{v}_1), q_2) \cdots (q_{n-1}, (a_{n-1}, \vec{v}_{n-1}), q_n)$$

of $\mathcal{A}$. We say that ρ *processes* the word $a_0 a_1 \cdots a_{n-1} \in \Sigma^*$, i.e., the $\vec{v}_j$ are ignored, and that ρ's extended Parikh image is $\sum_{j=0}^{n-1} \vec{v}_j$. The run is accepting, if it is either empty and both the initial state of $\mathcal{A}$ is accepting and the zero vector (the extended Parikh image of the empty run) is in C, or if it is nonempty, q_n is accepting, and ρ's extended Parikh image is in C. Finally, $(\mathcal{A}, C)$ accepts $w \in \Sigma^*$ if it has an accepting run processing w.

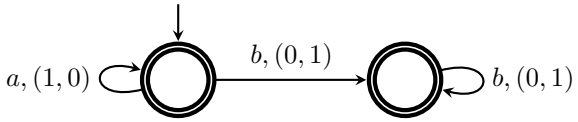

Figure 1 The automaton for Example 2.

▶ **Example 2.** Consider the deterministic PA $(\mathcal{A}, C)$ with $\mathcal{A}$ in Figure 1 and $C = \{(n, n) \mid n \in \mathbb{N}\} \cup \{(n, 2n) \mid n \in \mathbb{N}\}$. It accepts the language $\{a^n b^n \mid n \in \mathbb{N}\} \cup \{a^n b^{2n} \mid n \in \mathbb{N}\}$.

A cycle is a nonempty finite run infix

$$(q_0, w_0, q_1)(q_1, w_1, q_2) \cdots (q_{n-1}, w_{n-1}, q_n)(q_n, w_n, q_0)$$

starting and ending in the same state and such that the q_j are pairwise different. Note that every run infix containing at least n transitions contains a cycle, where n is the number of states of the automaton. Many of our proofs rely on the following shifting argument, which has been used before to establish inexpressibility results for Parikh automata [3].

▶ Remark 3. Let $\rho_0\rho_1\rho_2\rho_3$ be a run of a PA such that ρ_1 and ρ_3 are cycles starting in the same state. Then, $\Phi_e(\rho_0\rho_1\rho_2\rho_3) = \Phi_e(\rho_0\rho_2\rho_1\rho_3) = \Phi_e(\rho_0\rho_1\rho_3\rho_2)$. Furthermore, all three runs end in the same state and visit the same set of states (but maybe in different orders).

3 Parikh Automata over Infinite Words

In this section, we introduce Parikh automata over infinite words by lifting safety, reachability, Büchi, and co-Büchi acceptance from finite automata to Parikh automata. Recall that a Parikh automaton on finite words accepts if the last state of the run is accepting and the extended Parikh image of the run is in the semilinear set, i.e., both events are synchronized. For reachability and Büchi acceptance it is natural to consider both a synchronous and an asynchronous variant while for safety and co-Büchi there is only a synchronous variant.

All these automata have the same format as Parikh automata on finite words, but are now processing infinite words. Formally, consider $(\mathcal{A}, C)$ with $\mathcal{A} = (Q, \Sigma \times D, q_I, \Delta, F)$. Fix an infinite run $(q_0, w_0, q_1)(q_1, w_1, q_2)(q_2, w_2, q_3) \cdots$ of $\mathcal{A}$ with $q_0 = q_I$ (recall that each w_j is in $\Sigma \times D$), which we say *processes* $p_\Sigma(w_0 w_1 w_2 \cdots)$.

- The run is *safety accepting* if $\Phi_e(w_0 \cdots w_{n-1}) \in C$ and $q_n \in F$ for all $n \geqslant 0$.
- The run is *synchronous reachability accepting* if $\Phi_e(w_0 \cdots w_{n-1}) \in C$ and $q_n \in F$ for some $n \geqslant 0$.
- The run is *asynchronous reachability accepting* if $\Phi_e(w_0 \cdots w_{n-1}) \in C$ for some $n \geqslant 0$ and $q_{n'} \in F$ for some $n' \geqslant 0$.
- The run is *synchronous Büchi accepting* if $\Phi_e(w_0 \cdots w_{n-1}) \in C$ and $q_n \in F$ for infinitely many $n \geqslant 0$.
- The run is *asynchronous Büchi accepting* if $\Phi_e(w_0 \cdots w_{n-1}) \in C$ for infinitely many $n \geqslant 0$ and $q_{n'} \in F$ for infinitely many $n' \geqslant 0$.
- The run is *co-Büchi accepting* if there is an n_0 such that $\Phi_e(w_0 \cdots w_{n-1}) \in C$ and $q_n \in F$ for every $n \geqslant n_0$.

As mentioned before, we do not distinguish between synchronous and asynchronous co-Büchi acceptance, as these definitions are equivalent. Also, note that all our definitions are conjunctive in the sense that acceptance requires visits to accepting states *and* extended Parikh images in C. Thus, e.g., reachability and safety are not dual on a syntactic level. Nevertheless, we later prove dualities on a semantic level.

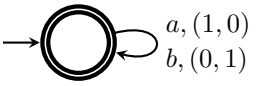

Figure 2 The automaton for Example 4.

Similarly, one can easily show that a disjunctive definition is equivalent to our conjunctive one: One can reflect in the extended Parikh image of a run prefix whether it ends in an accepting stateand then encode acceptance in the semilinear set. So, any given Parikh automaton $(\mathcal{A}, C)$ (with disjunctive or conjunctive acceptance) can be turned into another one $(\mathcal{A}', C')$ capturing acceptance in $(\mathcal{A}, C)$ by Parikh images only. So, with empty (full) set of accepting states and C' mimicking disjunction (conjunction), it is equivalent to the original automaton with disjunctive (conjunctive) acceptance.

Now, the language $L_{\mathrm{S}}(\mathcal{A}, C)$ of a safety Parikh automaton (SPA) $(\mathcal{A}, C)$ contains those words $w \in \Sigma^\omega$ such that $(\mathcal{A}, C)$ has a safety accepting run processing w. Similarly, we define the languages

- $L^s_{\mathrm{R}}(\mathcal{A}, C)$ of synchronous reachability Parikh automata (sRPA),
- $L^a_{\mathrm{R}}(\mathcal{A}, C)$ of asynchronous reachability Parikh automata (aRPA),
- $L^s_{\mathrm{B}}(\mathcal{A}, C)$ of synchronous Büchi Parikh automata (aBPA),
- $L^a_{\mathrm{B}}(\mathcal{A}, C)$ of asynchronous Büchi Parikh automata (sBPA), and
- $L_{\mathrm{C}}(\mathcal{A}, C)$ of co-Büchi Parikh automata (CPA).

Determinism for all types of automata is defined as for Parikh automata on finite words. Unless explicitly stated otherwise, every automaton is assumed to be nondeterministic.

▶ **Example 4.** Let $\mathcal{A}$ be the DFA shown in Figure 2 and let $C = \{(n, n) \mid n \in \mathbb{N}\}$ and $\overline{C} = \{(n, n') \mid n \neq n'\} = \mathbb{N}^2 \setminus C$.

Recall that a finite word w is (a, b)-balanced if $|w|_a = |w|_b$, i.e., the number of a's and b's in w is equal. The empty word is (a, b)-balanced and every odd-length word over $\{a, b\}$ is (a, b)-unbalanced.

1. When interpreting $(\mathcal{A}, C)$ as a PA, it accepts the language of finite (a, b)-balanced words; when interpreting $(\mathcal{A}, \overline{C})$ as a PA, it accepts the language of finite (a, b)-unbalanced words.
2. When interpreting $(\mathcal{A}, C)$ as an aRPA or sRPA, it accepts the language of infinite words that have an (a, b)-balanced prefix; when interpreting $(\mathcal{A}, \overline{C})$ as an aRPA or sRPA, it accepts the language of infinite words that have an (a, b)-unbalanced prefix. Note that both languages are universal, as the empty prefix is always (a, b)-balanced and every odd-length prefix is (a, b)-unbalanced.
3. When interpreting $(\mathcal{A}, C)$ as an SPA, it accepts the language of infinite words that have only (a, b)-balanced prefixes; when interpreting $(\mathcal{A}, \overline{C})$ as an SPA, it accepts the language of infinite words that have only (a, b)-unbalanced prefixes. Here, both languages are empty, which follows from the same arguments as for universality in the previous case.
4. When interpreting $(\mathcal{A}, C)$ as an aBPA or sBPA, it accepts the language of infinite words with infinitely many (a, b)-balanced prefixes; when interpreting $(\mathcal{A}, \overline{C})$ as an aBPA or sBPA, it accepts the language of infinite words with infinitely many (a, b)-unbalanced prefixes. The latter language is universal, as every odd-length prefix is unbalanced.
5. When interpreting $(\mathcal{A}, C)$ as a CPA, it accepts the language of infinite words such that almost all prefixes are (a, b)-balanced; when interpreting $(\mathcal{A}, \overline{C})$ as a CPA, it accepts the language of infinite words such that almost all prefixes are (a, b)-unbalanced. Again, the former language is empty.

Let $(\mathcal{A}, C)$ be a Parikh automaton. We say that a run prefix is an F-prefix if it ends in an accepting state of $\mathcal{A}$, a C-prefix if its extended Parikh image is in C, and an FC-prefix if it is both an F-prefix and a C-prefix. Note that both asynchronous acceptance conditions are defined in terms of the existence of F-prefixes and C-prefixes and all other acceptance conditions in terms of the existence of FC-prefixes.

▶ Remark 5. sRPA and aRPA (SPA, sBPA and aBPA, CPA) are strictly more expressive than ω-regular reachability (safety, Büchi, co-Büchi) automata. Inclusion follows by definition while strictness is witnessed by the languages presented in Example 4.

4 Expressiveness

In this section, we study the expressiveness of the various types of Parikh automata on infinite words introduced above, by comparing synchronous and asynchronous variants, deterministic and nondeterministic variants, and the different acceptance conditions.

▶ Remark 6. In this, and only this, section, we consider only reachability Parikh automata that are complete in the following sense: For every state q and every letter a there is a vector $\vec{v}$ and a state q' such that $(q, (a, \vec{v}), q')$ is a transition of $\mathcal{A}$, i.e., every letter can be processed from every state. Without this requirement, one can express safety conditions by incompleteness, while we want to study the expressiveness of "pure" reachability automata.

Safety, Büchi, and co-Büchi automata can be assumed, without loss of generality, to be complete, as one can always add a nonaccepting sink to complete such an automaton without modifying the accepted language.

We begin our study by comparing the synchronous and asynchronous variants of reachability and Büchi automata. All transformations proving the following inclusions are effective and lead to a linear increase in the number of states and a constant increase in the dimension of the semilinear sets.

▶ **Theorem 7.**

1. *aRPA and sRPA are equally expressive.*
2. *Deterministic aRPA are strictly more expressive than deterministic sRPA.*
3. *aBPA and sBPA are equally expressive.*
4. *Deterministic aBPA are strictly more expressive than deterministic sBPA.*

Due to the equivalence of synchronous and asynchronous (nondeterministic) reachability Parikh automata, we drop the qualifiers whenever possible and just speak of reachability Parikh automata (RPA). We do the same for (nondeterministic) Büchi Parikh automata (BPA).

Next, we compare the deterministic and nondeterministic variants for each acceptance condition. Note that all separations are as strong as possible, i.e., for reachability and Büchi we consider deterministic asynchronous automata, which are more expressive than their synchronous counterparts (see Theorem 7).

▶ **Theorem 8.**

1. *Nondeterministic RPA are strictly more expressive than deterministic aRPA.*
2. *Nondeterministic SPA are strictly more expressive than deterministic SPA.*
3. *Nondeterministic BPA are strictly more expressive than deterministic aBPA.*
4. *Nondeterministic CPA are strictly more expressive than deterministic CPA.*

After having separated deterministic and nondeterministic automata for all acceptance conditions, we now consider inclusions and separations between the different acceptance conditions. Here, the picture is notably different than in the classical ω-regular setting, as almost all classes can be separated.

▶ **Theorem 9.** *Every RPA can be turned into an equivalent BPA and into an equivalent CPA. All other automata types are pairwise incomparable.*

Our separations between the different acceptance conditions are as strong as possible, e.g., when we show that not every RPA has an equivalent SPA, we exhibit a *deterministic* sRPA (the weakest class of RPA) whose language is not accepted by any nondeterministic SPA (the strongest class of SPA). The same is true for all other separations.

5 Closure Properties

In this section, we study the closure properties of Parikh automata on infinite words. We begin by showing that, for deterministic synchronous automata, reachability and safety acceptance as well as Büchi and co-Büchi acceptance are dual, although they are not syntactically dual due to all acceptance conditions being defined by a conjunction. On the other hand, deterministic asynchronous automata can still be complemented, however only into nondeterministic automata.

▶ **Theorem 10.**

1. *Let $(\mathcal{A}, C)$ be a deterministic sRPA. The complement of $L_R^s(\mathcal{A}, C)$ is accepted by a deterministic SPA.*
2. *Let $(\mathcal{A}, C)$ be a deterministic aRPA. The complement of $L_R^a(\mathcal{A}, C)$ is accepted by an SPA, but not necessarily by a deterministic SPA.*
3. *Let $(\mathcal{A}, C)$ be a deterministic SPA. The complement of $L_S(\mathcal{A}, C)$ is accepted by a deterministic sRPA.*
4. *Let $(\mathcal{A}, C)$ be a deterministic sBPA. The complement of $L_B^s(\mathcal{A}, C)$ is accepted by a deterministic CPA.*
5. *Let $(\mathcal{A}, C)$ be a deterministic aBPA. The complement of $L_B^a(\mathcal{A}, C)$ is accepted by a CPA, but not necessarily by a deterministic CPA.*
6. *Let $(\mathcal{A}, C)$ be a deterministic CPA. The complement of $L_C(\mathcal{A}, C)$ is accepted by a deterministic sBPA.*

The positive results above are for deterministic automata. For nondeterministic automata, the analogous statements fail.

▶ **Theorem 11.**

1. *There exists an sRPA $(\mathcal{A}, C)$ such that no SPA accepts the complement of $L_R^s(\mathcal{A}, C)$.*
2. *There exists an SPA $(\mathcal{A}, C)$ such that no RPA accepts the complement of $L_S(\mathcal{A}, C)$.*
3. *There exists an sBPA $(\mathcal{A}, C)$ such that no CPA accepts the complement of $L_B^s(\mathcal{A}, C)$.*
4. *There exists a CPA $(\mathcal{A}, C)$ such that no BPA accepts the complement of $L_C(\mathcal{A}, C)$.*

Next, we consider closure under union, intersection, and complementation of the various classes of Parikh automata on infinite words. Notably, all nondeterministic (and some deterministic) classes are closed under union, the picture for intersection is more scattered, and we prove failure of complement closure for all classes. Again, this is in sharp contrast to the setting of classical Büchi automata, which are closed under all three Boolean operations.

▶ **Theorem 12.** *The closure properties depicted in Table 1 hold.*

Table 1 Closure properties and decidability of decision problems for Parikh automata on infinite words.

	Closure			Decision Problems			
	$\cup$	$\cap$	$\overline{\ }$	Emptiness	Universality	Model Check.	Games
RPA	✓	✓	✗	NP-compl.	undec.	undec.	undec.
det. aRPA	✗	✗	✗	NP-compl.	undec.	undec.	undec.
det. sRPA	✓	✗	✗	NP-compl.	undec.	undec.	undec.
SPA	✓	✓	✗	undec.	undec.	undec.	undec.
det. SPA	✗	✓	✗	undec.	coNP-compl.	coNP-compl.	undec.
BPA	✓	✗	✗	NP-compl.	undec.	undec.	undec.
det. aBPA	?	✗	✗	NP-compl.	undec.	undec.	undec.
det. sBPA	✓	✗	✗	NP-compl.	undec.	undec.	undec.
CPA	✓	✓	✗	undec.	undec.	undec.	undec.
det. CPA	✗	✓	✗	undec.	coNP-compl.	coNP-compl.	undec.

Note that there is one question mark in the closure properties columns in Table 1, which we leave for further research.

6 Decision Problems

In this section, we study the complexity of the nonemptiness and the universality problem, model checking, and solving games for Parikh automata on infinite words. Before we can do so, we need to specify how a Parikh automaton $(\mathcal{A}, C)$ is represented as input for algorithms: The vectors labeling the transitions of $\mathcal{A}$ are represented in binary and a linear set

$$\left\{ \vec{v}_0 + \sum_{i=1}^{k} c_i \vec{v}_i \;\middle|\; c_i \in \mathbb{N} \text{ for } i = 1, \ldots, k \right\}$$

is represented by the list $(\vec{v}_0, \ldots, \vec{v}_k)$ of vectors, again encoded in binary. A semilinear set is then represented by a set of such lists.

6.1 Nonemptiness

We begin by settling the complexity of the nonemptiness problem. The positive results are obtained by reductions to the emptiness of Parikh automata on finite words while the undecidability results are reductions from the termination problem for two-counter machines.

▶ **Theorem 13.** *The following problems are NP-complete:*

1. *Given an RPA, is its language nonempty?*
2. *Given a BPA, is its language nonempty?*

The following problems are undecidable:

3. *Given a deterministic SPA, is its language nonempty?*
4. *Given a deterministic CPA, is its language nonempty?*

Proof.

1. Due to Theorem 7.1, we only consider the case of sRPA for the NP upper bound. Given such an automaton $(\mathcal{A}, C)$ with $\mathcal{A} = (Q, \Sigma \times D, q_I, \Delta, F)$ let $F' \subseteq F$ be the set of accepting states from which a cycle is reachable. Now, define $\mathcal{A}' = (Q, \Sigma \times D, q_I, \Delta, F')$. Then, we have $L^s_{\mathrm{R}}(\mathcal{A}, C) \neq \emptyset$ if and only if $L(\mathcal{A}', C) \neq \emptyset$ (i.e., we treat $(\mathcal{A}', C)$ as a PA), with the latter problem being in NP [12].

The matching NP lower bound is, again due to Theorem 7.1, only shown for sRPA. We proceed by a reduction from the NP-complete [12] nonemptiness problem for Parikh automata. Given a Parikh automaton $(\mathcal{A}, C)$, let $\mathcal{A}'$ be obtained from $\mathcal{A}$ by adding a fresh state q with a self-loop labeled by $(\#, \vec{0})$ as well as transitions labeled by $(\#, \vec{0})$ leading from the accepting states of $\mathcal{A}$ to q. Here, $\#$ is a fresh letter and $\vec{0}$ is the zero vector of the correct dimension. By declaring q to be the only accepting state in $\mathcal{A}'$, we have that $L^s_{\mathrm{R}}(\mathcal{A}', C)$ is nonempty if and only if $L(\mathcal{A}, C)$ is nonempty.

Note that hardness holds already for deterministic automata, as one can always rename letters to make a nondeterministic PA deterministic without changing the answer to the emptiness problem.

2. Due to Theorem 7.3, it is enough to consider synchronous Büchi acceptance for the upper bound. So, fix some sBPA $(\mathcal{A}, C)$ with $\mathcal{A} = (Q, \Sigma \times D, q_I, \Delta, F)$. Let $C = \bigcup_i L_i$ where the L_i are linear sets. The language $L^s_{\mathrm{B}}(\mathcal{A}, C)$ is nonempty if and only if $L^s_{\mathrm{B}}(\mathcal{A}, L_i)$ is nonempty for some i. Hence, we show how to solve emptiness for automata with linear C, say $C = \left\{ \vec{v}_0 + \sum_{i=1}^k c_i \vec{v}_i \;\middle|\; c_i \in \mathbb{N} \text{ for } i = 1, \ldots, k \right\}$. We define $P = \left\{ \sum_{i=1}^k c_i \vec{v}_i \;\middle|\; c_i \in \mathbb{N} \text{ for } i = 1, \ldots, k \right\}$ and, for a given state $q \in Q$, the NFA

- $\mathcal{A}_q$ obtained from $\mathcal{A}$ by replacing the set of accepting states by $\{q\}$, and
- $\mathcal{A}_{q,q}$ obtained from $\mathcal{A}$ by replacing the initial state by q, by replacing the set of accepting states by $\{q\}$, and by modifying the resulting NFA such that it does not accept the empty word (but leaving its language unchanged otherwise).

We claim that $L^s_{\mathrm{B}}(\mathcal{A}, C)$ is nonempty if and only if there is a $q \in F$ such that both $L(\mathcal{A}_q, C)$ and $L(\mathcal{A}_{q,q}, P)$ are nonempty. As emptiness of Parikh automata is in NP, this yields the desired upper bound.

So, assume there is such a q. Then, there is a finite run ρ_1 of $\mathcal{A}$ that starts in q_I, ends in q, and processes some $w_1 \in \Sigma^*$ with extended Parikh image in C. Also, there is a finite run ρ_2 of $\mathcal{A}$ that starts and ends in q and processes some nonempty $w_2 \in \Sigma^*$ with extended Parikh image in P. For every $n \geqslant 1$, $\rho_1(\rho_2)^n$ is a finite run of $(\mathcal{A}, C)$ ending in the accepting state q that processes $w_1(w_2)^n$ and whose extended Parikh image is in C. So, $\rho_1(\rho_2)^\omega$ is a synchronous Büchi accepting run of $(\mathcal{A}, C)$.

For the converse direction, assume that there is some synchronous Büchi accepting run $(q_0, w_0, q_1)(q_1, w_1, q_2)(q_2, w_2, q_3)\cdots$ of $(\mathcal{A}, C)$. Then, there is also an accepting state $q \in F$ and an infinite set of positions $S \subseteq \mathbb{N}$ such that $q_s = q$ and $\Phi_e(w_0 \cdots w_{s-1}) \in C$ for all $s \in S$. Hence, for every $s \in S$ there is a vector $(c^s_1, \ldots, c^s_k) \in \mathbb{N}^k$ such that $\Phi_e(w_0 \cdots w_{s-1}) = \vec{v}_0 + \sum_{i=1}^k c^s_i \vec{v}_i$. By Dickson's Lemma [10], there are $s_1 < s_2$ such that $c^{s_1}_j \leq c^{s_2}_j$ for every $1 \leq j \leq k$. Then, $\Phi_e(w_{s_1} \cdots w_{s_2-1}) = \sum_{i=1}^k (c^{s_2}_i - c^{s_1}_i)\vec{v}_i$, which implies $\Phi_e(w_{s_1} \cdots w_{s_2-1}) \in P$. Thus, the prefix of ρ of length s_1 is an accepting run of the PA $(\mathcal{A}_q, C)$ and the next $(s_2 - s_1)$ transitions of ρ form a nonempty accepting run of the PA $(\mathcal{A}_{q,q}, P)$.

The NP lower bound follows from the proof of Theorem 13.1 by noticing that we also have that $L^s_{\mathrm{B}}(\mathcal{A}', C)$ is nonempty if and only if $L(\mathcal{A}, C)$ is nonempty.

3. and 4. The two undecidability proofs, based on reductions from undecidable problems for two-counter machines, are relegated to the appendix, which introduces all the required technical details on such machines. ◀

We conclude Subsection 6.1 by displaying an interesting consequence of the decomposition used in the proof of Theorem 13.2: we show that the language of every BPA (on infinite words) can be expressed by combining the languages of well-chosen PA (on finite words). This is similar to what happens in other settings: For instance, ω-regular languages are exactly the languages of the form $\bigcup_{j=1}^{n} L_j \cdot (L'_j)^\omega$, where each L_j, L'_j is regular. Analogously, ω-context-free languages can be characterized by context-free languages [7]. For Büchi Parikh automata, one direction of the above characterization holds:

▶ **Lemma 14.** *If a language L is accepted by a BPA then $L = \bigcup_{j=1}^{n} L_j \cdot (L'_j)^\omega$, where each L_j, L'_j is accepted by some PA.*

Proof. Let L be accepted by an sBPA $(\mathcal{A}, C)$ with $C = \bigcup_{j \in J} C_j$ for some finite set J, where each C_j is a linear set. In the proof of Theorem 13.2, we have defined the NFA $\mathcal{A}_q$ and $\mathcal{A}_{q,q}$ for every state q of $\mathcal{A}$.

Now, consider some $w \in L$ and an accepting run $\rho = (q_0, w_0, q_1)(q_1, w_1, q_2)(q_2, w_2, q_3)\cdots$ of $(\mathcal{A}, C)$ processing w. Then, there is an accepting state q of $\mathcal{A}$, some C_j, and an infinite set $S \subseteq \mathbb{N}$ of positions such that $q_s = q$ and $\Phi_e(w_0 \cdots w_{s-1}) \in C_j$ for all $s \in S$. Let $C_j = \left\{ \vec{v}_0 + \sum_{i=1}^{k} c_i \vec{v}_i \;\middle|\; c_i \in \mathbb{N} \text{ for } i = 1, \ldots, k \right\}$ and let $P_j = \left\{ \sum_{i=1}^{k} c_i \vec{v}_i \;\middle|\; c_i \in \mathbb{N} \text{ for } i = 1, \ldots, k \right\}$. As before, for every $s \in S$, there is a vector $(c_1^s, \ldots, c_k^s) \in \mathbb{N}^k$ such that $\Phi_e(w_0 \cdots w_{s-1}) = \vec{v}_0 + \sum_{i=1}^{k} c_i^s \vec{v}_i$.

Now, we apply an equivalent formulation of Dickson's Lemma [10], which yields an infinite subset $S' \subseteq S$ with $c_j^s \leq c_j^{s'}$ for all $1 \leq j \leq k$ and all $s, s' \in S'$ with $s < s'$, i.e., we have an increasing chain in S. Let $s_0 < s_1 < s_2 < \cdots$ be an enumeration of S'.

As above, $\Phi_e(w_{s_n} \cdots w_{s_{n+1}-1}) \in P_j$ for all n. So, $(q_0, w_0, q_1) \cdots (q_{s_0-1}, w_{s_0-1}, q_{s_0})$ is an accepting run of the PA $(\mathcal{A}_q, C)$ and each $(q_{s_n}, w_{s_n}, q_{s_n+1}) \cdots (q_{s_{n+1}-1}, w_{s_{n+1}-1}, q_{s_{n+1}})$ is an accepting run of the PA $(\mathcal{A}_{q,q}, P)$. So, $w \in \bigcup_{j \in J} \bigcup_{q \in Q} L(\mathcal{A}_q, C_j) \cdot (L(\mathcal{A}_{q,q}, P_j))^\omega$. ◀

Recall that a word is ultimately periodic if it is of the form xy^ω. Every nonempty ω-regular and every nonempty ω-context-free language contains an ultimately periodic word, which is a simple consequence of them being of the form $\bigcup_{j=1}^{n} L_j \cdot (L'_j)^\omega$.

▶ **Corollary 15.** *Every nonempty language accepted by a BPA contains an ultimately periodic word.*

Let us briefly comment on the other direction of the implication stated in Lemma 14, i.e., is every language of the form $\bigcup_{j=1}^{n} L_j \cdot (L'_j)^\omega$, where each L_j, L'_j is accepted by some PA, also accepted by some BPA? The answer is no: Consider $L = \{a^n b^n \mid n > 1\}$, which is accepted by a deterministic PA. However, using the shifting technique (see Remark 3), one can show that L^ω is not accepted by any BPA: Every accepting run of an n-state BPA processing $(a^n b^n)^\omega$ can be turned into an accepting run on a word of the form $(a^n b^n)^* a^{n+k} b^n (a^n b^n)^* a^{n-k} b^n (a^n b^n)^\omega$ for some $k > 0$ by shifting some cycle to the front while preserving Büchi acceptance.

For reachability acceptance, a similar characterization holds, as every RPA can be turned into an equivalent BPA. But for safety and co-Büchi acceptance the characterization question is nontrivial, as for these acceptance conditions all (almost all) run prefixes have to be FC-prefixes. We leave this problem for future work.

6.2 Universality

Now, we consider the universality problem. Here, the positive results follow from the duality of deterministic SPA and RPA (CPA and BPA) and the decidability of nonemptiness for the dual automata classes. Similarly, the undecidability proofs for deterministic sRPA and sBPA follow from duality and undecidability of nonemptiness for the dual automata classes. Finally, the remaining undecidability results follow from reductions from undecidable problems for two-counter machines and Parikh automata over finite words.

▶ **Theorem 16.** *The following problems are coNP-complete:*
1. *Given a deterministic SPA, is its language universal?*
2. *Given a deterministic CPA, is its language universal?*

The following problems are undecidable:
3. *Given a deterministic sRPA, is its language universal?*
4. *Given an SPA, is its language universal?*
5. *Given a deterministic sBPA, is its language universal?*
6. *Given a CPA, is its language universal?*

Proof. The proofs of the results for deterministic automata follow immediately from the fact that a language is universal if and only if its complement is empty, Theorem 10, and Theorem 13. The proof of undecidability for SPA, based on a reduction from the termination problem for two-counter machines, is relegated to the appendix where all the necessary technical details are presented. To conclude, let us consider universality of CPA.

6.) Universality of Parikh automata over finite words is undecidable [19]. Now, given a PA $(\mathcal{A}, C)$ over Σ, one can construct a CPA $(\mathcal{A}', C')$ for the language $L(\mathcal{A}, C) \cdot \# \cdot (\Sigma \cup \{\#\})^\omega \cup \Sigma^\omega$, where $\# \notin \Sigma$ is a fresh letter. This construction relies on freezing the counters (i.e., moving to a copy of the automaton with the same transition structure, but where the counters are no longer updated) and closure of CPA under union. Now, $L(\mathcal{A}, C)$ is universal if and only if $L_{\mathrm{C}}(\mathcal{A}', C')$ is universal. ◀

6.3 Model Checking

Model checking is arguably the most successful application of automata theory to automated verification. The problem asks whether a given system satisfies a specification, often given by an automaton.

More formally, and for the sake of notational convenience, we say that a transition system $\mathcal{T}$ is a (possibly incomplete) SPA $(\mathcal{A}, C)$ so that every state of $\mathcal{A}$ is accepting and $C = \mathbb{N}^d$, i.e., every run is accepting. Now, the model-checking problem for a class $\mathcal{L}$ of languages of infinite words asks, given a transition system $\mathcal{T}$ and a language $L \in \mathcal{L}$, whether $L_{\mathrm{S}}(\mathcal{T}) \subseteq L$, i.e., whether every word in the transition system satisfies the specification L. Note that our definition here is equivalent to the standard definition of model checking of finite-state transition systems.

Here, we study the model-checking problem for different types of Parikh automata.

▶ **Theorem 17.** *The following problems are coNP-complete:*
1. *Given a transition system $\mathcal{T}$ and a deterministic SPA $(\mathcal{A}, C)$, is $L_S(\mathcal{T}) \subseteq L_S(\mathcal{A}, C)$?*
2. *Given a transition system $\mathcal{T}$ and a deterministic CPA $(\mathcal{A}, C)$, is $L_S(\mathcal{T}) \subseteq L_C(\mathcal{A}, C)$?*

The following problems are undecidable:
3. *Given a transition system $\mathcal{T}$ and a deterministic sRPA $(\mathcal{A}, C)$, is $L_S(\mathcal{T}) \subseteq L^s_R(\mathcal{A}, C)$?*
4. *Given a transition system $\mathcal{T}$ and an SPA $(\mathcal{A}, C)$, is $L_S(\mathcal{T}) \subseteq L_S(\mathcal{A}, C)$?*
5. *Given a transition system $\mathcal{T}$ and a deterministic sBPA $(\mathcal{A}, C)$, is $L_S(\mathcal{T}) \subseteq L^s_B(\mathcal{A}, C)$?*
6. *Given a transition system $\mathcal{T}$ and a CPA $(\mathcal{A}, C)$, is $L_S(\mathcal{T}) \subseteq L_C(\mathcal{A}, C)$?*

Proof. Let $\mathcal{T}$ be a transition system with $L_S(\mathcal{T}) = \Sigma^\omega$, e.g., a one-state transition system with a self-loop labeled with all letters in Σ. Then, $L \subseteq \Sigma^\omega$ is universal if and only if L $L_S(\mathcal{T}) \subseteq L$. Thus, all six lower bounds (coNP-hardness and undecidability) immediately follow from the analogous lower bounds for universality (see Theorem 16). So, it remains to consider the two coNP upper bounds.

So, fix a deterministic SPA $(\mathcal{A}, C)$ and a transition system $\mathcal{T}$. We apply the usual approach to automata-theoretic model checking: We have $L_S(\mathcal{T}) \subseteq L_S(\mathcal{A}, C)$ if and only if $L_S(\mathcal{T}) \cap \overline{L_S(\mathcal{A}, C)} = \emptyset$. Due to Theorem 10.3 there is a deterministic sRPA $(\mathcal{A}', C')$ accepting $\overline{L_S(\mathcal{A}, C)}$. Furthermore, using a product construction, one can construct an RPA $(\mathcal{A}'', C'')$ accepting $L_S(\mathcal{T}) \cap \overline{L_S(\mathcal{A}, C)}$, which can then be tested for emptiness, which is in coNP (see Theorem 13). Note that the product construction depends on the fact that every run of $\mathcal{T}$ is accepting, i.e., the acceptance condition of the product automaton only has to check one acceptance condition.

The proof for deterministic co-Büchi Parikh automata is analogous, but using Büchi automata $(\mathcal{A}', C')$ and $(\mathcal{A}'', C'')$. ◀

6.4 Infinite Games

In this section, we study infinite games with winning conditions specified by Parikh automata. Such games are the technical core of the synthesis problem, the problem of determining whether there is a reactive system satisfying a given specification on its input-output behavior. Our main result is that solving infinite games is undecidable for all acceptance conditions we consider here.

Here, we consider Gale-Stewart games [15], abstract games induced by a language L of infinite words, in which two players alternately pick letters, thereby constructing an infinite word w. One player aims to ensure that w is in L while the other aims to ensure that it is not in L. Formally, given a language $L \subseteq (\Sigma_1 \times \Sigma_2)^\omega$, the game $G(L)$ is played between Player 1 and Player 2 in rounds $i = 0, 1, 2, \ldots$ as follows: At each round i, first Player 1 plays a letter $a_i \in \Sigma_1$ and then Player 2 answers with a letter $b_i \in \Sigma_2$. A play of $G(L)$ is an infinite outcome $w = \binom{a_0}{b_0}\binom{a_1}{b_1}\cdots$ and Player 2 wins it if and only if $w \in L$.

A strategy for Player 2 in $G(L)$ is a mapping from Σ_1^+ to Σ_2 that gives for each prefix played by Player 1 the next letter to play. An outcome $\binom{a_0}{b_0}\binom{a_1}{b_1}\cdots$ agrees with a strategy σ if for each i, we have that $b_i = \sigma(a_0 a_1 \ldots a_i)$. Player 2 wins $G(L)$ if she has a strategy that only agrees with outcomes that are winning for Player 2.

The next result follows immediately from the fact that for all classes of deterministic Parikh automata, either nonemptiness or universality is undecidable, and that these two problems can be reduced to solving Gale-Stewart games.

▶ **Theorem 18.** *The problem "Given an automaton $(\mathcal{A}, C)$, does Player 2 win $G(L(\mathcal{A}, C))$?" is undecidable for the following classes of automata: (deterministic) sRPA, (deterministic) SPA, (deterministic) sBPA, and (deterministic) CPA.*

Proof. The results follow immediately from the following two facts and the undecidability of emptiness or universality for the corresponding automata types. Fix a language L.

- Player 2 wins $G(\binom{\#}{L})$ with $\binom{\#}{L} = \{\binom{\#}{w_0}\binom{\#}{w_1}\binom{\#}{w_2}\cdots \mid w_0 w_1 w_2 \cdots \in L\}$ if and only if L is nonempty.
- Player 2 wins $G(\binom{L}{\#})$ with $\binom{L}{\#} = \{\binom{w_0}{\#}\binom{w_1}{\#}\binom{w_2}{\#}\cdots \mid w_0 w_1 w_2 \cdots \in L\}$ if and only if L is universal.

To conclude, note that a Parikh automaton for L can be turned into an equivalent one for $\binom{\#}{L}$ and $\binom{L}{\#}$ while preserving determinism and the acceptance type, by just replacing each transition label a by $\binom{\#}{a}$ and $\binom{a}{\#}$, respectively. ◀

7 Conclusion

In this work, we have extended Parikh automata to infinite words and studied expressiveness, closure properties, and decision problems. Unlike their ω-regular counterparts, Parikh automata on infinite words do not form a nice hierarchy induced by their acceptance conditions. This is ultimately due to the fact that transitions cannot be disabled by the counters running passively along a run. Therefore, a safety condition on the counters cannot be turned into a, say, Büchi condition on the counters, something that is trivial for state conditions. Furthermore, we have shown that emptiness, universality, and model checking are decidable for some of the models we introduced, but undecidable for others. Most importantly, we prove coNP-completeness of model checking with specifications given by deterministic Parikh automata with safety and co-Büchi acceptance. This allows for the automated verification of quantitative safety and persistence properties. Finally, solving infinite games is undecidable for all models.

Note that we have "only" introduced reachability, safety, Büchi, and co-Büchi Parikh automata. There are many more acceptance conditions in the ω-regular setting, e.g., parity, Rabin, Streett, and Muller. We have refrained from generalizing these, as any natural definition of these acceptance conditions will subsume co-Büchi acceptance, and therefore have an undecidable nonemptiness problem.

In future work, we aim to close the open closure property in Table 1. Also, we leave open the complexity of the decision problems in case the semilinear sets are not given by their generators, but by a Presburger formula.

One of the appeals of Parikh automata over finite words is their robustness: they can equivalently be defined via a quantitative variant of WMSO, via weighted automata, and other models (see the introduction for a more complete picture). In future work, we aim to provide similar alternative definitions for Parikh automata on infinite words, in particular, comparing our automata to blind multi-counter automata [11] and reversal-bounded counter machines [18]. However, let us mention that the lack of closure properties severely limits the chances for a natural fragment of MSO being equivalent to Parikh automata on infinite words.

Let us conclude with the following problem for further research: If a Parikh automaton with, say safety acceptance, accepts an ω-regular language, is there then an equivalent ω-regular safety automaton? Stated differently, does Parikhness allow to accept more ω-regular languages? The same question can obviously be asked for other acceptance conditions as well.

References

1 Vincent D. Blondel, Olivier Bournez, Pascal Koiran, Christos H. Papadimitriou, and John N. Tsitsiklis. Deciding stability and mortality of piecewise affine dynamical systems. *Theor. Comput. Sci.*, 255(1-2):687–696, 2001. doi:10.1016/S0304-3975(00)00399-6.

2 Alin Bostan, Arnaud Carayol, Florent Koechlin, and Cyril Nicaud. Weakly-unambiguous Parikh automata and their link to holonomic series. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *ICALP 2020*, volume 168 of *LIPIcs*, pages 114:1–114:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.114.

3 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. On the expressiveness of Parikh automata and related models. In Rudolf Freund, Markus Holzer, Carlo Mereghetti, Friedrich Otto, and Beatrice Palano, editors, *NCMA 2011*, volume 282 of *books@ocg.at*, pages 103–119. Austrian Computer Society, 2011.

4 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Unambiguous constrained automata. *Int. J. Found. Comput. Sci.*, 24(7):1099–1116, 2013. doi:10.1142/S0129054113400339.

5 Giusi Castiglione and Paolo Massazza. On a class of languages with holonomic generating functions. *Theor. Comput. Sci.*, 658:74–84, 2017. doi:10.1016/j.tcs.2016.07.022.

6 Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Regular Separability of Parikh Automata. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP 2017*, volume 80 of *LIPIcs*, pages 117:1–117:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.117.

7 Rina S. Cohen and Arie Y. Gold. Theory of omega-languages. I. Characterizations of omega-context-free languages. *J. Comput. Syst. Sci.*, 15(2):169–184, 1977. doi:10.1016/S0022-0000(77)80004-4.

8 Luc Dartois, Emmanuel Filiot, and Jean-Marc Talbot. Two-way Parikh automata with a visibly pushdown stack. In Mikolaj Bojanczyk and Alex Simpson, editors, *FOSSACS 2019*, volume 11425 of *LNCS*, pages 189–206. Springer, 2019. doi:10.1007/978-3-030-17127-8_11.

9 Jürgen Dassow and Victor Mitrana. Finite automata over free groups. *Int. J. Algebra Comput.*, 10(6):725–738, 2000. doi:10.1142/S0218196700000315.

10 Leonard E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Amer. Journal Math.*, 35(4):413–422, 1913.

11 Henning Fernau and Ralf Stiebe. Blind counter automata on omega-words. *Fundam. Informaticae*, 83(1-2):51–64, 2008. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi83-1-2-06.

12 Diego Figueira and Leonid Libkin. Path logics for querying graphs: Combining expressiveness and efficiency. In *LICS 2015*, pages 329–340. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.39.

13 Emmanuel Filiot, Shibashis Guha, and Nicolas Mazzocchi. Two-way Parikh automata. In Arkadev Chattopadhyay and Paul Gastin, editors, *FSTTCS 2019*, volume 150 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.40.

14 Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin. A pattern logic for automata with outputs. *Int. J. Found. Comput. Sci.*, 31(6):711–748, 2020. doi:10.1142/S0129054120410038.

15 David Gale and F. M. Stewart. Infinite games with perfect information. In Harold William Kuhn and Albert William Tucker, editors, *Contributions to the Theory of Games (AM-28), Volume II*, chapter 13, pages 245–266. Princeton University Press, 1953.

16 Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966. doi:pjm/1102994974.

17 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. Parikh automata over infinite words. *arXiv*, 2207.07694, 2022. doi:10.48550/arXiv.2207.07694.

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

19 Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *ICALP 2003*, volume 2719 of *LNCS*, pages 681–696. Springer, 2003. doi:10.1007/3-540-45061-0_54.

20 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

21 Victor Mitrana and Ralf Stiebe. Extended finite automata over groups. *Discret. Appl. Math.*, 108(3):287–300, 2001. doi:10.1016/S0166-218X(00)00200-6.

22 J. Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In Ernest Nagel, Patrick Suppes, and Alfred Tarski, editors, *Logic, Methodology and Philosophy of Science*, volume 44 of *Studies in Logic and the Foundations of Mathematics*, pages 1–11. Elsevier, 1966. doi:10.1016/S0049-237X(09)70564-6.

23 Karianto Wong. Parikh automata with pushdown stack, 2004. Diploma thesis, RWTH Aachen University. URL: https://old.automata.rwth-aachen.de/download/papers/karianto/ka04.pdf.

A Appendix: Parikh Automata and Two-counter Machines

Many of our undecidability proofs are reductions from nontermination problems for two-counter machines. To simulate these machines by Parikh automata, we require them to be in a certain normal form. We first introduce two-counter machines, then the normal form, and conclude this section by presenting the simulation via Parikh automata.

A two-counter machine $\mathcal{M}$ is a sequence

$$(0 : \mathtt{I}_0)(1 : \mathtt{I}_1) \cdots (k-2 : \mathtt{I}_{k-2})(k-1 : \mathtt{STOP}),$$

where the first element of a pair $(\ell : \mathtt{I}_\ell)$ is the line number and $\mathtt{I}_\ell$ for $0 \leq \ell < k-1$ is an instruction of the form

- $\mathtt{INC(X}_i\mathtt{)}$ with $i \in \{0,1\}$,
- $\mathtt{DEC(X}_i\mathtt{)}$ with $i \in \{0,1\}$, or
- $\mathtt{IF\ X}_i\mathtt{=0\ GOTO}\ \ell'\ \mathtt{ELSE\ GOTO}\ \ell''$ with $i \in \{0,1\}$ and $\ell', \ell'' \in \{0, \ldots, k-1\}$.

A configuration of $\mathcal{M}$ is of the form (ℓ, c_0, c_1) with $\ell \in \{0, \ldots, k-1\}$ (the current line number) and $c_0, c_1 \in \mathbb{N}$ (the current contents of the counters). The initial configuration is $(0,0,0)$ and the unique successor configuration of a configuration (ℓ, c_0, c_1) is defined as follows:

- If $\mathtt{I}_\ell = \mathtt{INC(X}_i\mathtt{)}$, then the successor configuration is $(\ell+1, c_0', c_1')$ with $c_i' = c_i + 1$ and $c_{1-i}' = c_{1-i}$.
- If $\mathtt{I}_\ell = \mathtt{DEC(X}_i\mathtt{)}$, then the successor configuration is $(\ell+1, c_0', c_1')$ with $c_i' = \max\{c_i - 1, 0\}$ and $c_{1-i}' = c_{1-i}$.
- If $\mathtt{I}_\ell = \mathtt{IF\ X}_i\mathtt{=0\ GOTO}\ \ell'\ \mathtt{ELSE\ GOTO}\ \ell''$ and $c_i = 0$, then the successor configuration is (ℓ', c_0, c_1).
- If $\mathtt{I}_\ell = \mathtt{IF\ X}_i\mathtt{=0\ GOTO}\ \ell'\ \mathtt{ELSE\ GOTO}\ \ell''$ and $c_i > 0$, then the successor configuration is (ℓ'', c_0, c_1).
- If $\mathtt{I}_\ell = \mathtt{STOP}$, then (ℓ, c_0, c_1) has no successor configuration.

The unique run of $\mathcal{M}$ (starting in the initial configuration) is defined as expected. It is either finite (line $k-1$ is reached) or infinite (line $k-1$ is never reached). In the former case, we say that $\mathcal{M}$ terminates.

▶ **Proposition 19** ([20]). *The following problem is undecidable: Given a two-counter machine $\mathcal{M}$, does $\mathcal{M}$ terminate?*

In the following, we assume without loss of generality that each two-counter machine satisfies the *guarded-decrement property*: Every decrement instruction $(\ell : \mathtt{DEC(X}_i\mathtt{)})$ is preceded by $(\ell-1 : \mathtt{IF\ X}_i\mathtt{=0\ GOTO}\ \ell+1\ \mathtt{ELSE\ GOTO}\ \ell)$ and decrements are never the target of a goto instruction. As the decrement of a zero counter has no effect, one can modify each two-counter machine $\mathcal{M}$ into an $\mathcal{M}'$ satisfying the guarded-decrement property such that $\mathcal{M}$ terminates if and only if $\mathcal{M}'$ terminates: One just adds the the required guard before every decrement instruction and changes each target of a goto instruction that is a decrement instruction to the preceding guard.

The guarded-decrement property implies that decrements are only executed if the corresponding counter is nonzero. Thus, the value of counter i after a finite sequence of executed instructions (starting with value zero in the counters) is equal to the number of executed increments of counter i minus the number of executed decrements of counter i. Note that the number of executed increments and decrements can be tracked by a Parikh automaton.

Consider a finite or infinite word $w = w_0 w_1 w_2 \cdots$ over the set $\{0, 1, \ldots, k-1\}$ of line numbers. We now describe how to characterize whether w is (a prefix of) the projection to the line numbers of the unique run of $\mathcal{M}$ starting in the initial configuration. This

characterization is designed to be checkable by a Parikh automaton. Note that w only contains line numbers, but does not encode values of the counters. These will be kept track of by the Parikh automaton by counting the number of increment and decrement instructions in the input, as explained above (this explains the need for the guarded-decrement property). Formally, we say that w contains an *error* at position $n < |w| - 1$ if either $w_n = k - 1$ (the instruction in line w_n is `STOP`), or if one of the following two conditions is satisfied:

1. The instruction $\mathtt{I}_{w_n}$ in line w_n of $\mathcal{M}$ is an increment or a decrement and $w_{n+1} \neq w_n + 1$, i.e., the letter w_{n+1} after w_n is not equal to the line number $w_n + 1$, which it should be after an increment or decrement.
2. $\mathtt{I}_{w_n}$ has the form `IF` $\mathtt{X}_i$`=0 GOTO` ℓ `ELSE GOTO` ℓ', and one of the following cases holds: Either, we have

$$\sum\nolimits_{j\colon \mathtt{I}_j = \mathtt{INC}(\mathtt{X}_i)} |w_0 \cdots w_n|_j = \sum\nolimits_{j\colon \mathtt{I}_j = \mathtt{DEC}(\mathtt{X}_i)} |w_0 \cdots w_n|_j$$

and $w_{n+1} \neq \ell$, i.e., the number of increments of counter i is equal to the number of decrements of counter i in $w_0 \cdots w_n$ (i.e., the counter is zero) but the next line number in w is not the target of the if-branch. Or, we have

$$\sum\nolimits_{j\colon \mathtt{I}_j = \mathtt{INC}(\mathtt{X}_i)} |w_0 \cdots w_n|_j \neq \sum\nolimits_{j\colon \mathtt{I}_j = \mathtt{DEC}(\mathtt{X}_i)} |w_0 \cdots w_n|_j,$$

and $w_{n+1} \neq \ell'$, i.e., the number of increments of counter i is not equal to the number of decrements of counter i in $w_0 \cdots w_n$ (i.e., the counter is nonzero) but the next line number in w is not the target of the else-branch.

Note that the definition of error (at position n) refers to the number of increments and decrements in the prefix $w_0 \cdots w_n$, which does not need to be error-free itself. However, if a sequence of line numbers does not have an error, then the guarded-decrement property yields the following result.

▶ **Lemma 20.** *Let $w \in \{0, 1, \ldots, k-1\}^+$ with $w_0 = 0$. Then, w has no errors at positions $\{0, 1, \ldots, |w| - 2\}$ if and only if w is a prefix of the projection to the line numbers of the run of $\mathcal{M}$.*

Proof. If w has no errors at positions $\{0, 1, \ldots, |w| - 2\}$, then an induction shows that (w_n, c_0^n, c_1^n) with

$$c_i^n = \sum\nolimits_{j\colon \mathtt{I}_j = \mathtt{INC}(\mathtt{X}_i)} |w_0 \cdots w_{n-1}|_j - \sum\nolimits_{j\colon \mathtt{I}_j = \mathtt{DEC}(\mathtt{X}_i)} |w_0 \cdots w_{n-1}|_j$$

is the n-th configuration of the run of $\mathcal{M}$.

On the other hand, projecting a prefix of the run of $\mathcal{M}$ to the line numbers yields a word w without errors at positions $\{0, 1, \ldots, |w| - 2\}$. ◀

The existence of an error can be captured by a Parikh automaton, leading to the undecidability of the safe word problem for Parikh automata, which we now prove. Let $(\mathcal{A}, C)$ be a PA accepting finite words over Σ. A *safe* word of $(\mathcal{A}, C)$ is an infinite word in Σ^ω such that each of its prefixes is in $L(\mathcal{A}, C)$.

▶ **Lemma 21.** *The following problem is undecidable: Given a deterministic PA, does it have a safe word?*

Proof. Our proof proceeds by a reduction from the nontermination problem for decrement-guarded two-counter machines. Given such a machine $\mathcal{M} = (0 : \mathtt{I}_0) \cdots (k-2 : \mathtt{I}_{k-2})(k-1 : \mathtt{STOP})$ let $\Sigma = \{0, \ldots, k-1\}$ be the set of its line numbers. We construct a deterministic PA $(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ that accepts a word $w \in \Sigma^*$ if and only if $w = \varepsilon$, $w = 0$, or if $|w| \geqslant 2$ and w

does not contain an error at position $|w|-2$ (but might contain errors at earlier positions). Intuitively, the automaton checks whether the second-to-last instruction is executed properly. The following is then a direct consequence of Lemma 20: $(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ has a safe word if and only if $\mathcal{M}$ does not terminate.

The deterministic PA $(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ keeps track of the occurrence of line numbers with increment and decrement instructions of each counter (using four dimensions) and two auxiliary dimensions to ensure that the two cases in Condition 2 of the error definition on Page 17 are only checked when the second-to-last letter corresponds to a goto instruction. More formally, we construct $\mathcal{A}_\mathcal{M}$ such the unique run processing some input $w = w_0 \dots w_{n-1}$ has the extended Parikh image $(v^0_{\text{inc}}, v^0_{\text{dec}}, v^0_{\text{goto}}, v^1_{\text{inc}}, v^1_{\text{dec}}, v^1_{\text{goto}})$ where

- v^i_{inc} is equal to $\sum_{j\colon \mathtt{I}_j=\mathtt{INC(X}_i\mathtt{)}} |w_0 \cdots w_{n-2}|_j$, i.e., the number of increment instructions read so far (ignoring the last letter),
- v^i_{dec} is equal to $\sum_{j\colon \mathtt{I}_j=\mathtt{DEC(X}_i\mathtt{)}} |w_0 \cdots w_{n-2}|_j$, i.e., the number of decrement instructions read so far (ignoring the last letter), and
- $v^i_{\text{goto}} \bmod 4 = 0$, if the second-to-last instruction $\mathtt{I}_{w_{n-2}}$ is not a goto testing counter i,
- $v^i_{\text{goto}} \bmod 4 = 1$, if the second-to-last instruction $\mathtt{I}_{w_{n-2}}$ is a goto testing counter i and the last letter w_{n-1} is equal to the target of the if-branch of this instruction, and
- $v^i_{\text{goto}} \bmod 4 = 2$, if the second-to-last instruction $\mathtt{I}_{w_{n-2}}$ is a goto testing counter i and the last letter w_{n-1} is equal to the target of the else-branch of this instruction.
- $v^i_{\text{goto}} \bmod 4 = 3$, if the second-to-last instruction $\mathtt{I}_{w_{n-2}}$ is a goto testing counter i and the last letter w_{n-1} is neither equal to the target of the if-branch nor equal to the target of the else-branch of this instruction. Note that this constitutes an error at position $n-2$.

Note that $v^i_{\text{goto}} \bmod 4 \neq 0$ can be true for at most one of the i at any time (as a goto instruction $\mathtt{I}_{w_{n-2}}$ only refers to one counter) and that the v^i_{inc} and v^i_{dec} are updated with a delay of one transition (as the last letter of w is ignored). This requires to store the previously processed letter in the state space of $\mathcal{A}_\mathcal{M}$.

Further, $C_\mathcal{M}$ is defined such that $(v^0_{\text{inc}}, v^0_{\text{dec}}, v^0_{\text{goto}}, v^1_{\text{inc}}, v^1_{\text{dec}}, v^1_{\text{goto}})$ is in $C_\mathcal{M}$ if and only if

- $v^i_{\text{goto}} \bmod 4 = 0$ for both i, or if
- $v^i_{\text{goto}} \bmod 4 = 1$ for some i (recall that i is unique then) and $v^i_{\text{inc}} = v^i_{\text{dec}}$, or if
- $v^i_{\text{goto}} \bmod 4 = 2$ for some i (again, i is unique) and $v^i_{\text{inc}} \neq v^i_{\text{dec}}$.

All other requirements, e.g., Condition 1 of the error definition on Page 17, the second-to-last letter not being $k-1$, and the input being in $\{\varepsilon, 0\}$, can be checked using the state space of $\mathcal{A}_\mathcal{M}$. ◀

A.1 Proofs omitted in Section 6

First, we prove that nonemptiness for deterministic SPA is undecidable.

Proof of Theorem 13.3. The result follows immediately from Lemma 21: A PA $(\mathcal{A}, C)$ has a safe word if and only if $L_\text{S}(\mathcal{A}, C) \neq \emptyset$. ◀

Next, we prove undecidability of nonemptiness for deterministic CPA.

Proof of Theorem 13.4. We present a reduction from the universal termination problem [1][1] for decrement-guarded two-counter machines, which is undecidable. The problem asks whether a given two-counter machine $\mathcal{M}$ terminates from every configuration. If this is the case, we say that $\mathcal{M}$ is universally terminating.

[1] The authors use a slightly different definition of two-counter machine than we do here. Nevertheless, the universal termination problem for their machines can be reduced to the universal termination problem for decrement-guarded two-counter machines as defined here.

Now, consider a decrement-guarded two-counter machine $\mathcal{M}$ that contains, without loss of generality, an increment instruction for each counter, say in lines ℓ_0^+ and ℓ_1^+. In the proof of Lemma 21, we construct a deterministic PA $(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ that accepts a finite word w over the line numbers of $\mathcal{M}$ if and only if $w = \varepsilon$, $w = 0$, or if $|w| \geqslant 2$ and w does not contain an error at position $|w| - 2$. We claim that $L_\mathrm{C}(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ is nonempty if and only if $\mathcal{M}$ is not universally terminating.

So, first assume there is some $w \in L_\mathrm{C}(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$. Hence, there is a run of $(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ processing w satisfying the co-Büchi acceptance condition: From some point n_0 onward, the run only visits states in F and the extended Parikh image is in $C_\mathcal{M}$. This means that there is no error in w after position n_0. So, $\mathcal{M}$ does not terminate from the configuration (w_{n_0}, c_0, c_1) with

$$c_i = \sum\nolimits_{j\colon \mathtt{I}_j = \mathtt{INC(X}_i\mathtt{)}} |w_0 \cdots w_{n_0-1}|_j - \sum\nolimits_{j\colon \mathtt{I}_j = \mathtt{DEC(X}_i\mathtt{)}} |w_0 \cdots w_{n_0-1}|_j,$$

i.e., $\mathcal{M}$ is not universally terminating.

Now, assume $\mathcal{M}$ does not universally terminate, say it does not terminate from configuration (ℓ, c_0, c_1). Recall that the instruction in line ℓ_i^+ is an increment of counter i. We define $w = (\ell_0^+)^{c_0}(\ell_1^+)^{c_1} w'$ where w' is the projection to the line numbers of the (nonterminating) run of $\mathcal{M}$ starting in (ℓ, c_0, c_1). This word does not have an error after position ℓ (but may have some before that position). Hence there is a co-Büchi accepting run of $(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ processing w, i.e., $L_\mathrm{C}(\mathcal{A}_\mathcal{M}, C_\mathcal{M})$ is nonempty. ◀

Finally, we prove undecidability of universality for SPA.

Proof of Theorem 16.4. We present a reduction from the termination problem for (decrement-guarded) two-counter machines. So, fix such a machine $\mathcal{M}$ with line numbers $0, 1, \ldots, k-1$ where $k-1$ is the line number of the stopping instruction, fix $\Sigma = \{0, 1, \ldots, k-1\}$, and consider $L_\mathcal{M} = L_\mathcal{M}^0 \cup L_\mathcal{M}^1$ with

$$\begin{aligned} L_\mathcal{M}^0 =& \{w \in \Sigma^\omega \mid w_0 \neq 0\} \text{ and} \\ L_\mathcal{M}^1 =& \{w \in \Sigma^\omega \mid \text{if } |w|_{k-1} > 0 \text{ then } w \text{ contains an error} \\ & \qquad\qquad \text{strictly before the first occurrence of } k-1\}. \end{aligned}$$

We first prove that $L_\mathcal{M}$ is not universal if and only if $\mathcal{M}$ terminates, then that $L_\mathcal{M}$ is accepted by some SPA.

So, assume that $\mathcal{M}$ terminates and let $w \in \Sigma^*$ be the projection of the unique finite run of $\mathcal{M}$ to the line numbers. Then, w starts with 0, contains a $k-1$, and no error before the $k-1$. Thus, $w0^\omega$ is not in $L_\mathcal{M}$, i.e., $L_\mathcal{M}$ is not universal.

Conversely, assume that $L_\mathcal{M}$ is not universal. Then, there is an infinite word w that is neither in $L_\mathcal{M}^0$ nor in $L_\mathcal{M}^1$. So, w must start with 0, contain a $k-1$, but no error before the first $k-1$. Thus, Lemma 20 implies that the prefix of w up to and including the first $k-1$ is the projection to the line numbers of the run of $\mathcal{M}$. This run is terminating, as the prefix ends in $k-1$. Thus, $\mathcal{M}$ terminates.

It remains to argue that $L_\mathcal{M} = L_\mathcal{M}^0 \cup L_\mathcal{M}^1$ is accepted by an SPA. Due to closure of SPA under union and the fact that every ω-regular safety property is also accepted by an SPA, we only need to consider $L_\mathcal{M}^1$. Recall that we have constructed a deterministic PA $(\mathcal{A}, C)$ (on finite words) accepting a word if it is empty, 0, or contains an error at the second-to-last position. We modify this PA into an SPA $(\mathcal{A}', C')$ that accepts $L_\mathcal{M}^1$.

To this end, we add a fresh accepting state q_a and a fresh rejecting state q_r to $\mathcal{A}$ while making all states of $\mathcal{A}$ accepting in $\mathcal{A}'$. Both fresh states are sinks equipped with self-loops that are labeled with $(\ell, \vec{0})$ for every line number ℓ. Here, $\vec{0}$ is the appropriate zero vector, i.e., the counters are frozen when reaching the fresh states.

Intuitively, moving to q_a signifies that an error at the current position is guessed. Consequently, if a $k-1$ is processed from a state of $\mathcal{A}$, the rejecting sink q_s is reached. We reflect in a fresh component of the extended Parikh image whether q_a has been reached. Due to this, we can define C' so that the extended Parikh image of every run prefix ending in a state of $\mathcal{A}$ is in C' and that the extended Parikh image of a run prefix ending in q_a is in C' if removing the last entry (the reflecting one) yields a vector in C, i.e., an error has indeed occurred. Then, we have $L_{\mathrm{S}}(\mathcal{A}', C') = L^1_{\mathcal{M}}$ as required. ◀