Counting and Sliding

Verifying and Restoring Healthy Systems

Dissertation

Mario Grobler, M. Sc.

Universität Bremen
Fachbereich 3 – Mathematik und Informatik

Datum des Kolloquiums

02. September 2024

Gutachter

Professor Dr. Sebastian Siebertz, Universität Bremen

Associate Professor Dr. Martin Zimmermann, Universität Aalborg

Abstract

Reactive systems play a significant role in the daily life of every person. Such systems include personal computers, automatic teller machines, devices we use every day in our households, and also systems that play a less active but still important role, for example monitoring devices scanning for environmental threats in a country. In particular for safety-critical reactive systems it is of enormous importance that correctness properties of such systems have been verified. A very prominent technique to tackle such verification tasks is called *model checking*. From a theoretical point of view, model checking commonly boils down to solving classical decision problems for finite automata; usually automata operating on infinite words.

Still, due the dynamics of the real world it might be the case that a change in the environment leads to a faulty or even dangerous state of such a system. Referring to the above example of devices scanning for environmental threats, assume that due to a mistake at a local construction site the connection to the other devices has been cut, leaving a part of the country unmonitored. Hence, a new solution has to be computed. However, the new solution can differ arbitrarily from the current state. Given that it is costly to move such devices around, it is desirable to compute a new solution that is close to the current state. To capture and formalize such scenarios, we introduce the new framework of *solution discovery via reconfiguration* for constructing a feasible solution for a given problem by executing a sequence of small modifications starting from a given state.

This thesis consists of two independent parts. In the first part, we study Parikh automata on finite and infinite words with applications to model checking. Parikh automata extend classical finite automata with a Counting mechanism. Several variants of Parikh automata operating on infinite words were recently introduced. We show that one of these variants coincides with blind counter automata on infinite words. Every ω-language recognized by a blind counter automata is of the form $\bigcup_i U_i V_i^\omega$ for Parikh recognizable languages U_i, V_i, but blind counter machines fall short of characterizing this class of ω-languages. We introduce several additional variants of Parikh automata on infinite words that yield automata characterizations of classes of ω-language of the form $\bigcup_i U_i V_i^\omega$ for all combinations of languages U_i, V_i being regular or Parikh-recognizable. When both U_i and V_i are regular, this coincides with Büchi's classical theorem. We study the effect of ε-transitions in all variants of Parikh automata and show that almost all of them admit ε-elimination. Furthermore, we investigate the deterministic variants of the newly introduced models, compare the

expressiveness of all models and conclude by studying their classical decision problems with applications to model checking.

In the second part, we introduce the new framework of *solution discovery via reconfiguration* and exemplify our framework on a multitude of fundamental graph problems. The solution discovery variant of a graph problem consists of an input graph where tokens are initially placed on the vertices or edges. Every token can be moved in a predetermined way for a cost of one, for example, by Sliding them to a neighboring vertex or edge. Then the question is whether we can discover an arbitrary solution for our problem starting with the initial token placement within a given budget. We study the classical as well as the parameterized complexity of the solution discovery variants of several graph problems and explore the boundary between tractable and intractable instances.

Zusammenfassung

Reaktive Systeme spielen eine signifikante Rolle im täglichen Leben jeder Person. Beispiele solcher Systeme sind PCs, Geldautomaten, Haushaltsgeräte und auch Systeme, die eine weniger aktive, aber dennoch wichtige Rolle spielen, beispielsweise Überwachungssysteme, die ein Land nach Umweltbedrohungen absuchen. Insbesondere für sicherheitskritische Systeme ist es von erheblicher Wichtigkeit, dass deren Korrektheit verifiziert wurde. Eine sehr prominente Technik um an solche Verifizierungsaufgaben heranzugehen heißt *model checking*. Aus theoretischer Sicht lässt sich model checking häufig auf die Lösung klassischer Entscheidungsprobleme für endliche Automaten reduzieren; in der Regel Automaten, die auf unendlichen Wörtern laufen.

Dennoch kann es passieren, dass aufgrund eines Umwelteinflusses ein System einen fehlerhaften oder sogar gefährlichen Zustand einnimmt. Bezugnehmend auf das oben genannte Beispiel eines Überwachungssystems für Umweltbedrohungen, kann es passieren, dass aufgrund eines Versehens bei einer naheliegenden Baustelle die Verbindung eines Überwachungssystems gekappt wird, wodurch ein Teil des Landes unüberwacht ist. Somit ist die Neuberechnung einer Lösung erforderlich. Diese neue Lösung kann sich jedoch beliebig vom aktuellen Zustand unterscheiden. Unter der Annahme, dass es teuer ist ein solches Gerät zu einer neuen Position zu bewegen, ist die Berechnung einer Lösung erwünscht, die dem aktuellen Zustand möglichst nahe kommt. Um solche Szenarien zu formalisieren, führen wir das neue Framework *solution discovery via reconfiguration* ein. Somit können wir gültige Lösungen für ein gegebenes Problem konstruieren, indem wir eine Sequenz von kleinen Modifikationen, beginnend in einem gegebenen Zustand, ausführen.

Diese Arbeit besteht aus zwei unabhängigen Teilen. Im ersten Teil untersuchen wir Parikh-Automaten auf endlichen und unendlichen Wörtern mit model checking Anwendungen. Parikh-Automaten erweitern klassische endliche Automaten mit einem Mechanismus zum Zählen. Diverse Varianten von Parikh-Automaten, die auf unendlichen Wörtern laufen, wurden kürzlich vorgestellt. Wir zeigen, dass eines dieser Modelle die gleiche Ausdrucksstärke wie Blindzählerautomaten auf unendlichen Wörtern hat. Jede ω-Sprache, die von einem Blindzählerautomaten erkannt wird, ist von der Form $\bigcup_i U_i V_i^\omega$ für Parikh-erkennbare Sprachen U_i, V_i. Dennoch sind sie nicht mächtig genug, um diese Klasse von ω-Sprachen zu charakterisieren. Wir führen diverse weitere Varianten von Parikh-Automaten auf unendlichen Wörtern ein, die Charakterisierungen für Klassen von ω-Sprachen der Form $\bigcup_i U_i V_i^\omega$ für alle Kombinationen von regulären bzw. Parikh-erkennbaren U_i, V_i liefern. Wenn

sowohl U_i als auch V_i regulär sind, erhalten wir genau den Satz von Büchi. Wir untersuchen den Einfluss von ε-Transitionen in allen Varianten von Parikh-Automaten und zeigen, dass fast alle Modelle ε-Elimination zulassen. Darüber hinaus untersuchen wir die deterministischen Varianten der neu eingeführten Modelle, vergleichen deren Ausdrucksstärke, und schließen mit einer Untersuchung der klassischen Entscheidungsprobleme mit Anwendungen zum model checking ab.

Im zweiten Teil führen wir das neue Framework *solution discovery via reconfiguration* ein und veranschaulichen unser Framework an einer Vielzahl von fundamentalen Graphproblemen. Die Discovery-Variante eines Graphproblems besteht aus einem Eingabegraphen, auf dessen Knoten oder Kanten bereits Token platziert sind. Jedes Token kann in einer vorbestimmten Weise mit Kosten 1 bewegt werden, zum Beispiel durch Schieben zu einem benachbarten Knoten bzw. einer benachbarten Kante. Dann stellt sich die Frage, ob wir eine beliebige Lösung unseres Problems im Rahmen eines gegebenen Budgets entdecken können, wenn wir mit der gegebenen Tokenplatzierung starten. Wir studieren sowohl die klassische Komplexität als auch die parametrisierte Komplexität der Discovery-Varianten diverser Graphprobleme und untersuchen die Grenze zwischen effizient lösbaren und nicht effizient lösbaren Instanzen.

Preamble

This thesis consists of two independent parts. We recall the relevant notions and definitions from the first part in the second part accordingly.

In the first part, we study Parikh automata on finite and infinite words. First we establish some results for Parikh automata on finite words. Following, we present several definitions of Parikh automata on infinite words. We consider the deterministic as well as the nondeterministic variants and study closure properties, expressiveness, and common decision problems with applications to model checking. Furthermore, we compare our models to other models with counting mechanisms operating on infinite words.

The content of the first part is based on the following publications. We note that some results in Chapter 2 have not been published. The author of this thesis contributed to a significant majority in both of these works.

[GSS24] Mario Grobler, Leif Sabellek and Sebastian Siebertz. *Remarks on Parikh-recognizable omega-languages.* Presented at CSL 2024.

[GS24] Mario Grobler and Sebastian Siebertz. *Deterministic Parikh automata on infinite words.* Submitted to CSL 2025.

We emphasize that Georg Zetzsche was involved in insightful discussions that eventually lead to some results in the first part. Thank you Georg!

In the second part, we introduce the new framework of *solution discovery via reconfiguration* motivated by the dynamics of real-world applications. We exemplify our framework on a multitude of fundamental graph problems, namely Vertex Cover, Independent Set, and Dominating Set, being classical NP-complete problems; as well as on Vertex Cut and Edge Cut, being classical problems in P.

The content of the second part is based on the following publications. The author of this thesis contributed to a substantial portion of the presented excerpts.

[FGM+23] Michael R. Fellows, Mario Grobler, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Frances A. Rosamond, Daniel Schmand and Sebastian Siebertz. *On Solution Discovery via Reconfiguration.* Presented at ECAI 2023.

[GMM+23] Mario Grobler, Stephanie Maaz, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Daniel Schmand, Sebastian Siebertz. *Solution discovery via reconfiguration for problems in* P. Accepted at ICALP 2024.

Contents

Part I

Parikh Automata on Finite and Infinite Words

1 Prelude

1.1 Introduction

Finite automata find numerous applications in formal language theory, logic, verification, and many more, in particular due to their good closure properties and algorithmic properties. To enrich this spectrum of applications even more, it has been a fruitful direction to add features to finite automata to capture also situations beyond the regular realm.

One such possible extension of finite automata with counting mechanisms has been introduced by Greibach in her study of blind and partially blind (one-way) multicounter machines [Gre78]. Blind multicounter machines are generalized by weighted automata as introduced in [MS01]. Parikh automata (PA) were introduced by Klaedtke and Ruess in [KR03b]. A Parikh automaton is a non-deterministic finite automaton that is additionally equipped with a semi-linear set C, and every transition is equipped with a d-tuple of non-negative integers. Whenever an input word is read, d counters are initialized with the values 0 and every time a transition is used, the counters are incremented by the values in the tuple of the transition accordingly. An input word is accepted if the PA ends in an accepting state and additionally, the resulting d-tuple of counter values lies in C. Klaedtke and Ruess showed that PA are equivalent to weighted automata over the group $(\mathbb{Z}^k, +, \mathbf{0})$, and hence equivalent to Greibach's blind multicounter machines, as well as to reversal bounded multicounter machines [BB74, Iba78]. Recently it was shown that these models can be translated into each other using only logarithmic space [BDG$^+$23]. In this work we call the class of languages recognized by any of these models *Parikh recognizable*. Klaedtke and Ruess [KR03b] showed that the class of Parikh recognizable languages is precisely the class of languages definable in weak existential monadic second-order logic of one successor extended with linear cardinality constraints. The class of Parikh recognizable languages contains all regular languages, but also many more, even languages that are not context-free, e. g., the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$. On the other hand, the language of palindromes is context-free, but not Parikh recognizable. On finite words, blind multicounter automata, Parikh automata and related models have been investigated extensively, extending [Gre78, KR03b] for example by affine PA and PA on letters [CFM11, CFM12a], bounded PA [CFM12b], two-way PA [FGM19], PA with a pushdown stack [Kar04] as well as a combination of both [DFT19], history-deterministic PA [EGJ$^+$23], automata and grammars with valences [FS02, Hoo02], and several algorithmic applications, e. g., in the context of path logics for querying graphs [FL15].

In the well-studied realm of verification of reactive systems, automata-related approaches provide a powerful framework to tackle important problems such as model checking problems [BK08, CGP99, CHVB18]. However, computations of such systems are generally represented as infinite objects, as we often expect them to not terminate (but rather interact with the environment). Hence, automata processing infinite words are well-suited to approach the model checking problem. One common approach is the following: assume we are given a system, e. g., represented as a Kripke structure $\mathcal{K}$, and a specification

represented as an automaton $\mathcal{A}$ (or any formalism that can be translated into one) accepting all correct computations. Then we can verify the correctness of the system by solving the inclusion problem, that is, answering the question whether every computation of $\mathcal{K}$ is also a computation of $\mathcal{A}$. However, solving the inclusion problem generally requires to complement the automaton $\mathcal{A}$, which is often expensive or not even possible. Hence, another common counterexample-driven approach is the following. Let $\mathcal{A}$ be an automaton accepting all incorrect computations. Then we can verify that the system has no incorrect computations by solving the intersection-emptiness problem of $\mathcal{K}$ and $\mathcal{A}$, that is, answering the question whether their sets of computations are disjoint. Büchi automata, recognizing ω-regular languages, provide a natural extension of classical (finite word) automata to infinite words, enjoying closure under the Boolean operations and decidable decision problems. However, being one of the most basic models, their expressiveness is quite limited.

Let us consider two examples. In a three-user setting in an operating system we would like to ensure that none of the users gets a lot more resources than the other two. One way to represent a such a specification is considering the ω-language

$$\{\alpha \in \{a,b,c\}^\omega \mid \text{ there are infinitely many prefixes } w \text{ of } \alpha \text{ with } |w|_a = |w|_b = |w|_c\},$$

stating that there are infinitely many moments where the resources are distributed equally. Similarly, one could provide a set of unwanted computations via the ω-language

$$\{\alpha \in \{a,b,c\}^\omega \mid \text{ there are infinitely many prefixes } w \text{ of } \alpha \text{ with } |w|_a > |w|_b + |w|_c\},$$

stating that one user gets more resources than the other two users combined infinitely often. As another example, consider a classical producer-consumer setting, where a producer continuously produces a good, and a consumer consumes these goods continuously. We can model this setting as an infinite word and ask that at no time the consumer has consumed more than the producer has produced at this time. Bad computations can be modeled via the ω-language

$$\{\alpha \in \{p,c\}^\omega \mid \text{there is a prefix } w \text{ of } \alpha \text{ with } |w|_c > |w|_p\}.$$

Such specifications are not ω-regular, as these require to "count arbitrarily". This motivates the study of finite automata with counting mechanisms on infinite words. Fernau and Stiebe initiated this study and introduced blind counter automata on infinite words [FS08], extending Greibachs blind multicounter machines. Independently, Klaedtke and Ruess proposed possible extensions of Parikh automata to infinite words. This line of research was recently picked up by Guha et al. [GJLZ22].

Guha et al. [GJLZ22] introduced *safety, reachability, Büchi- and co-Büchi Parikh automata*. These models provide natural generalizations of automata models with Parikh conditions on infinite words. One shortcoming of safety, reachability and co-Büchi Parikh automata is that they do not generalize Büchi automata, that is, they cannot recognize all ω-regular

languages. The non-emptiness problem, which is highly relevant for model checking applications, is undecidable for safety and co-Büchi Parikh automata. Furthermore, none of these models is closed under the ω-operation, meaning that for every model there is a Parikh recognizable (finite word) language L such that L^ω is not recognizable by any of these models. Guha et al. raised the question whether (appropriate variants of) Parikh automata on infinite words have the same expressiveness as blind counter automata on infinite words.

Büchi's famous theorem states that ω-regular languages are characterized as languages of the form $\bigcup_i U_i V_i^\omega$, where the U_i and V_i are regular languages [Bü60]. As a consequence of the theorem, many properties of ω-regular languages are inherited from regular languages. For example, algorithms deciding non-emptiness for finite word automata can be used to solve the non-emptiness problem for Büchi automata as well. In their systematic study of blind counter automata operating on infinite words, Fernau and Stiebe [FS08] considered the class $\mathcal{K}_*$, the class of ω-languages of the form $\bigcup_i U_i V_i^\omega$ for Parikh recognizable languages U_i and V_i. They proved that the class of ω-languages recognizable by blind counter automata is a proper subset of the class $\mathcal{K}_*$. They posed as an open problem to provide automata models that capture classes of ω-languages of the form $\bigcup_i U_i V_i^\omega$ where U_i and V_i are described by a certain mechanism. In this work we propose *reachability-regular Parikh automata*, *limit Parikh automata*, *strong reset Parikh automata*, and *weak reset Parikh automata* as new automata models and study their deterministic and non-deterministic variants. First, we focus on their non-deterministic variants.

We pick up the question of Fernau and Stiebe [FS08] to consider classes of ω-languages of the form $\bigcup_i U_i V_i^\omega$ where U_i and V_i are described by a certain mechanism. We define the four classes $\mathcal{L}^\omega_{\mathsf{Reg},\mathsf{Reg}}$, $\mathcal{L}^\omega_{\mathsf{PA},\mathsf{Reg}}$, $\mathcal{L}^\omega_{\mathsf{Reg},\mathsf{PA}}$ and $\mathcal{L}^\omega_{\mathsf{PA},\mathsf{PA}}$ of ω-languages of the form $\bigcup_i U_i V_i^\omega$, where the U_i, V_i are regular or Parikh recognizable languages of finite words, respectively. By Büchi's theorem the class $\mathcal{L}^\omega_{\mathsf{Reg},\mathsf{Reg}}$ is the class of ω-regular languages.

We show that the newly introduced (non-deterministic) reachability-regular Parikh automata, which are a small modification of reachability Parikh automata (as introduced by Guha et al. [GJLZ22]) capture exactly the class $\mathcal{L}^\omega_{\mathsf{PA},\mathsf{Reg}}$. This model turns out to be equivalent to (non-deterministic) limit Parikh automata. This model was hinted at in the concluding remarks of [KR03b].

Fully resolving the classification of the above mentioned classes we introduce weak and strong reset Parikh automata, whose non-deterministic variants turn out to be equivalent. In contrast to all other Parikh models, these are closed under the ω-operation, while maintaining all algorithmic properties of PA (in particular, non-emptiness is NP-complete and hence decidable). We show that the class of ω-languages recognized by reset PA is a strict superclass of $\mathcal{L}^\omega_{\mathsf{PA},\mathsf{PA}}$. We show that appropriate graph-theoretic restrictions of reset PA exactly capture the classes $\mathcal{L}^\omega_{\mathsf{PA},\mathsf{PA}}$ and $\mathcal{L}^\omega_{\mathsf{Reg},\mathsf{PA}}$, yielding the first automata characterizations for these classes. On our way, we study the closure properties of the newly introduced

models and obtain the following hierarchy.

$$\begin{aligned}\text{reachability PA} &\subsetneq \text{reachability-regular PA} = \text{limit PA}\\ &\subsetneq \text{Büchi PA} \subsetneq \text{weak reset PA} = \text{strong reset PA}.\end{aligned}$$

The automata models introduced by Guha et al. [GJLZ22] do not have ε-transitions, while blind counter automata as introduced by Fernau and Stiebe [FS08] have such transitions. Towards answering the question of Guha et al. we study the effect of ε-transitions in all Parikh automata models. We show that all models except safety and co-Büchi Parikh automata admit ε-elimination. This in particular answers the question of Guha et al. [GJLZ22] whether blind counter automata and Büchi Parikh automata have the same expressiveness on infinite words affirmative. We show that safety and co-Büchi PA with ε-transitions are strictly more powerful than their variants without ε-transitions. In particular, co-Büchi PA with ε-transitions generalize Büchi PA, while safety PA with ε-transitions are even more powerful than reset PA.

Finally, we study the classical decision problems for the newly introduced models, namely emptiness, membership and universality. We show that the results for (non-deterministic) PA on finite words translate to the infinite word setting, that is, emptiness is coNP-complete, membership is NP-complete and universality is undecidable. Additionally, we study the intersection-emptiness and inclusion problems of the newly introduced models, as these are important problems in order to study model checking problems, the core problems in the field of formal verification. Formally, we are given a system K as a Kripke structure (a safety automaton) or a PA and a specification $\mathcal{A}$ as a PA. The question whether at least one computation of a Kripke structure satisfies the specification (which we call existential safety model checking and boils down to solving intersection-emptiness) is motivated by the testing the absence of incorrect computations, as described above. Similarly, the question whether all computations of a PA satisfy the specification (which we call universal PA model checking and boils down to solving inclusion) has been studied in [GJLZ22] for reachability PA, Büchi PA, safety PA and co-Büchi PA. Guha et al. [GJLZ22] show that this problem is undecidable for the non-deterministic variants of these models. We study this problem as well as the universal safety model checking problem and the existential PA model checking problem for the remaining deterministic models. We refer to Table 1.1, Table 1.2, and Table 1.3 for an overview of the results.

As indicated above, there are many scenarios where non-determinism adds expressiveness or succinctness to their deterministic counterparts. However, this often comes at the price of important decision problems becoming hard to solve, or even undecidable, see also [CHVB18, GTW02]. This is also the case for Parikh automata, as witnessed, e. g., by co-Büchi PA and safety PA. As mentioned above, Guha et al. [GJLZ22] have shown that universality is undecidable for the non-deterministic variants of these models, yet being more powerful than their deterministic counterparts. However, the deterministic variants enjoy a coNP-complete (and hence decidable) universality problem [GJLZ22].

	$\cup$	$\cap$	$\overline{}$
limit PA	✓	✓	✗
reachability-regular PA	✓	✓	✗
weak reset PA	✓	✗	✗
strong reset PA	✓	✗	✗
deterministic limit PA	✓	✓	✓
deterministic reachability-regular PA	✗	✗	✗
deterministic weak reset PA	✗	✗	✗
deterministic strong reset PA	✗	✗	✗

Table 1.1. Closure properties. The bar in the right column denotes the complement.

This motivates the study of the deterministic variants of the newly introduced models, namely deterministic limit PA, deterministic reachability-regular PA, deterministic strong reset PA, and deterministic weak reset PA. We investigate their expressiveness, closure properties, and common decision problems. First we show that the above mentioned hierarchy results for the non-deterministic variants do not translate to the deterministic setting. While

$$\text{deterministic strong reset PA} \subsetneq \text{deterministic weak reset PA}$$

and

$$\begin{aligned}&\text{deterministic reachability PA}\\ &\qquad\subsetneq \text{ deterministic reachability-regular PA}\\ &\qquad\subsetneq \text{deterministic weak reset PA}\end{aligned}$$

still holds, all other models become pairwise incomparable. Furthermore, we show that among all studied deterministic models only deterministic limit PA generalize Büchi automata in the sense that they recognize all ω-regular languages.

Deterministic limit PA also shine in the light of closure properties: as we show, among all studied PA operating on infinite words (the deterministic variants as well as the non-deterministic ones) they are the only ones closed under all Boolean operations.

This benefit also yields decidable decision problems. In contrast to the all other models that were mentioned before, emptiness and universality are both decidable for deterministic limit PA. We show that also strong reset PA benefit from determinism: although having bad closure properties, their universality problems becomes decidable. However, as we show,

	$L = \emptyset$?	$uv^\omega \in L$?	$L = \Sigma^\omega$?
limit PA	coNP-complete	NP-complete	undecidable
reachability-regular PA	coNP-complete	NP-complete	undecidable
weak reset PA	coNP-complete	NP-complete	undecidable
strong reset PA	coNP-complete	NP-complete	undecidable
det. limit PA	coNP-complete	NP-complete	decidable, Π_2^{P}-hard
det. reachability-regular PA	coNP-complete	NP-complete	undecidable
det. weak reset PA	coNP-complete	NP-complete	undecidable
det. strong reset PA	coNP-complete	NP-complete	Π_2^{P}-complete

Table 1.2. Decision problems.

for deterministic reachability-regular PA and deterministic weak reset PA the universality problem remains undecidable.

While universality and hence the universal model checking problems are undecidable for all non-deterministic variants of the aforementioned models, the situation changes for deterministic limit PA and deterministic strong reset PA. Again, we refer to Table 1.1, Table 1.2, and Table 1.3 for an overview of the results.

We remark that the Π_2^{P}-hardness results we obtain for deterministic limit PA are not tight, that is, it remains open whether these problems can be solved in Π_2^{P}. However, we are able to show that if we have the guarantee that the semi-linear sets of the PA can be complemented in polynomial time, then the Π_2^{P}-hard problems for deterministic limit PA and deterministic strong reset PA become coNP-complete. We also remark that the coNP-completeness result for deterministic Büchi PA does not follow from [GJLZ22].

	Model Checking			
	Kripke		PA	
	∃	∀	∃	∀
limit PA	coNP-c.	undec.	coNP-c.	undec.
reachability-regular PA	coNP-c.	undec.	coNP-c.	undec.
weak reset PA	coNP-c.	undec.	undec.	undec.
strong reset PA	coNP-c.	undec.	undec.	undec.
reachability PA	coNP-c. (□)	undec. (□)	coNP-c. (□)	undec. (□)
Büchi PA	coNP-c.	undec. (□)	coNP-c.	undec. (□)
safety PA	undec. (□)	undec. (□)	undec. (□)	undec. (□)
co-Büchi PA	undec. (□)	undec. (□)	undec. (□)	undec. (□)
det. limit PA	coNP-c.	dec., Π_2^P-hard	coNP-c.	dec., Π_2^P-hard
det. reachability-regular PA	coNP-c.	undec.	coNP-c.	undec.
det. weak reset PA	coNP-c.	undec.	undec.	undec.
det. strong reset PA	coNP-c.	Π_2^P-c.	undec.	Π_2^P-c.
det. reachability PA	coNP-c. (□)	undec. (□)	coNP-c. (□)	undec. (□)
det. Büchi PA	coNP-c.	undec. (□)	coNP-c.	undec. (□)
det. safety PA	undec. (□)	coNP-c. (□)	undec. (□)	coNP-c. (□)
det. co-Büchi PA	undec. (□)	coNP-c. (□)	undec. (□)	coNP-c. (□)

Table 1.3. (Un)decidability results of model checking problems. The entries marked with a (□) were shown in (or follow immediately from) [GJLZ22]

1.2 Preliminaries

In this section we present the definitions and notations that are relevant for the first part of the thesis. We write $\mathbb{Z}$ for the set of all integers and $\mathbb{N}$ for the set of non-negative integers including 0. For $m, n \in \mathbb{N}$, we denote by $[m, n]$ the set $\{m, m+1, \ldots, n\}$ with the convention that $[m, n] = \emptyset$ if $m > n$. We abbreviate the set $[1, n]$ by $[n]$. Furthermore, let $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$. Throughout this part we mainly use the variables $c, d, i, j, k, \ell, m, n, z$ to denote positive integers and we will tacitly assume this if not explicitly stated otherwise.

1.2.1 Words and Languages

Let Σ be an alphabet, i. e., a finite non-empty set and denote by Σ^* the set of all finite words over Σ. A language is a subset $L \subseteq \Sigma^*$. For a word $w \in \Sigma^*$, we denote by $|w|$ the length of w, and by $|w|_a$ the number of occurrences of the symbol $a \in \Sigma$ in w. We write ε for the empty word of length 0 and denote by $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ the set of all non-empty words over Σ. We say a word $v \in \Sigma^*$ is a *subword* of a word $w \in \Sigma^*$, denoted by $v \preccurlyeq w$, if v can be obtained from w by removing symbols at arbitrary positions of w. More formally, $v = v_1 \ldots v_m$ is a subword of $w = w_1 \ldots w_n$ if there is a function $f : [m] \to [n]$ such that $f(i) < f(j)$ if $i < j$ and $v_i = w_{f(i)}$ for all $i \leq m$. Similarly, we call v *prefix* of $w = w_1 \ldots w_n$ if $v = w_1 \ldots w_i$ for some $i \leq n$; *suffix* of w if $v = w_i \ldots w_n$ for some $i \leq n+1$; and *infix* of w if $v = w_i \ldots w_j$ for $i, j \leq n+1$. Note that ε is a prefix, suffix, and infix of every word $w \in \Sigma^*$.

An *infinite word* over an alphabet Σ is a function $\alpha : \mathbb{N} \setminus \{0\} \to \Sigma$. We often write α_i instead of $\alpha(i)$. Thus, we can understand an infinite word as an infinite sequence of symbols $\alpha = \alpha_1\alpha_2\alpha_3 \ldots$. Hence, the notions of prefix, infix and suffix translate to infinite words the obvious way; however, we only consider finite prefixes and infixes of infinite words, while every suffix is always an infinite word itself. For $m \leq n$, we abbreviate the infix $\alpha_m \ldots \alpha_n$ of α by $\alpha[m : n]$. We denote by Σ^ω the set of all infinite words over Σ. We call a subset $L \subseteq \Sigma^\omega$ an *ω-language*. Moreover, for $L \subseteq \Sigma^*$, we define $L^\omega = \{w_1 w_2 \cdots \mid w_i \in L \setminus \{\varepsilon\}\} \subseteq \Sigma^\omega$. We call an infinite word $\alpha \subseteq \Sigma^\omega$ *ultimately periodic* if it can be written as $\alpha = uv^\omega$ for finite words $u, v \in \Sigma^*$. Likewise, we call a non-empty ω-language *ultimately periodic* if it contains an ultimately periodic infinite word.

In addition to the convention of variable use mentioned above, we mainly use the variables a and b to denote symbols, w to denote finite words and α, β to denote infinite words and we will tacitly assume this if not explicitly stated otherwise.

1.2.2 Regular and ω-regular Languages

A *non-deterministic finite automaton* (NFA) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$, where Q is a finite set of states, Σ is the input alphabet, $q_0 \in Q$ is the initial state, $\Delta \subseteq Q \times \Sigma \times Q$ is the set of

transitions and $F \subseteq Q$ is the set of accepting states. We call $\mathcal{A}$ *deterministic* if for every pair $(p, a) \in Q \times \Sigma$ there is exactly one transition of the form $(p, a, q) \in \Delta$ for some $q \in Q$. A *run* of $\mathcal{A}$ on a word $w = w_1 \dots w_n \in \Sigma^*$ is a (possibly empty) sequence of transitions $r = r_1 \dots r_n$ with $r_i = (p_{i-1}, w_i, p_i) \in \Delta$ such that $p_0 = q_0$. We say r is *accepting* if $p_n \in F$. The empty run on ε is accepting if $q_0 \in F$. We define the *language recognized by* $\mathcal{A}$ as

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \text{there is an accepting run of } \mathcal{A} \text{ on } w\}.$$

If a language L is recognized by some NFA $\mathcal{A}$, we call L *regular*.

A *Büchi automaton* is an NFA $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ that takes infinite words as input. A *run* of $\mathcal{A}$ on an infinite word $\alpha_1\alpha_2\alpha_3 \dots$ is an infinite sequence of transitions $r = r_1 r_2 r_3 \dots$ with $r_i = (p_{i-1}, \alpha_i, p_i) \in \Delta$ such that $p_0 = q_0$. We say r is *accepting* if there are infinitely many i with $p_i \in F$. We define the *ω-language recognized by* $\mathcal{A}$ as $L_\omega(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid$ there is an accepting run of $\mathcal{A}$ on $\alpha\}$. If an ω-language L is recognized by some Büchi automaton $\mathcal{A}$, we call L *ω-regular*.

Büchi's theorem establishes an important connection between regular and ω-regular languages:

Theorem 1.2.1 (Büchi [Bü60])**.** *A language $L \subseteq \Sigma^\omega$ is ω-regular if and only if there are regular languages $U_1, V_1, \dots, U_n, V_n \subseteq \Sigma^*$ for some $n \geq 1$ such that $L = \bigcup_{i \leq n} U_i V_i^\omega$.*

Throughout this part we will denote the class of ω-regular languages by $\mathcal{L}^\omega_{\mathsf{Reg},\mathsf{Reg}}$.

If every state of a Büchi automaton $\mathcal{A}$ is accepting, we call $\mathcal{A}$ a *safety automaton*. Similarly, a *Muller automaton* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \mathcal{F})$, where Q, Σ, q_0, and Δ are defined as for Büchi automata, and $\mathcal{F} \subseteq 2^Q$ is a collection of sets of accepting states. Runs are defined as for Büchi automata, and a run r is accepting if the sets of states that appear infinitely often in r is contained in $\mathcal{F}$. Deterministic Muller automata have the same expressiveness as nondeterministic Büchi automata [McN66]. However, deterministic Büchi automata are less expressive than their non-deterministic counterpart [Lan69]. If an ω-language L is recognized by some deterministic Büchi automaton, we call L *deterministic ω-regular*. For a (finite word) language $W \subseteq \Sigma^*$, we define $\overline{W} = \{\alpha \in \Sigma^\omega \mid \alpha[1:i] \in W$ for infinitely many $i\}$. An ω-language L is deterministic ω-regular if and only if $L = \overline{W}$ for a regular language W [Lan69].

1.2.3 Pushdown Automata

A *pushdown automaton* (PDA) is a finite automaton equipped with a stack, that is, a first-in-last-out buffer where only the top-most element can be accessed. Formally, a PDA is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, Z_0, \Delta)$, where Q, Σ and q_0 are defined as for NFA. Additionally, Γ is the stack alphabet, Z_0 is the initial stack symbol, and $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\} \times \Gamma^+ \times \Gamma^* \times Q)$ is a finite set of transitions. Intuitively, a transition $(p, a, \eta, \zeta, q) \in \Delta$ means that if the PDA is currently

in state p and reads the symbol a while the top most stack symbols[1] yield the word η, the PDA may replace η by ζ and move to state q. This intuition is formalized using the notion of configurations. A *configuration* of $\mathcal{A}$ is a tuple $(q, w, \zeta) \in Q \times \Sigma^* \times \Gamma^*$, where q is the current state, w is the suffix of the input word that has not been read yet, and ζ is the current stack content. Let c, c' be two configurations of $\mathcal{A}$. We say c *derives* into c', written $c \vdash_{\mathcal{A}} c'$, if $c = (q, aw, \eta\xi), c' = (q', w, \zeta\xi)$ and $(q, a, \eta, \zeta, q') \in \Delta$; or if $c = (q, w, \eta\xi), c' = (q', w, \zeta\xi)$ and $(q, \varepsilon, \eta, \zeta, q') \in \Delta$. Note that ε-transitions are allowed, that is, we do not necessarily need to read a symbol of the input word to change the state or alter the stack content. A *computation* of $\mathcal{A}$ is a sequence of configurations $c_1, \ldots, c_n$ for some $n \geq 1$ with $c_i \vdash_{\mathcal{A}} c_{i+1}$ for all $1 \leq i < n$. We write $c \vdash^*_{\mathcal{A}} c'$ if there is a computation $c_1, \ldots, c_n$ of $\mathcal{A}$ with $c = c_1$ and $c_n = c'$. We define the language recognized by $\mathcal{A}$ as

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^*_{\mathcal{A}} (q, \varepsilon, \varepsilon) \text{ for a } q \in Q\}.$$

If a language L is recognized by some PDA $\mathcal{A}$, we call L *context-free*.

1.2.4 Semi-linear Sets and Presburger Arithmetic

A *linear set* (of dimension $d \geq 1$) is a set of the form

$$C(\mathbf{b}, P) = \{\mathbf{b} + \mathbf{p}_1 z_1 + \cdots + \mathbf{p}_\ell z_\ell \mid z_1, \ldots, z_\ell \in \mathbb{N}\} \subseteq \mathbb{N}^d,$$

where $\mathbf{b} \in \mathbb{N}^d$ and $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_\ell\} \subseteq \mathbb{N}^d$ is a finite set of vectors for some $\ell \geq 0$. We call $\mathbf{b}$ the *base vector* and P the set of *period vectors*. We call linear sets of the form $C(\mathbf{0}, P)$ *homogeneous*. A *semi-linear set* is a finite union of linear sets. We also consider semi-linear sets over $\mathbb{N}^d_\infty$, that is, semi-linear sets with an additional symbol ∞ for infinity. As usual, addition of vectors and multiplication of a vector with a number is defined component-wise, where $z + \infty = \infty + z = \infty + \infty = \infty$ for all $z \in \mathbb{N}$, $z \cdot \infty = \infty \cdot z = \infty$ for all $z \geq 1$, and $0 \cdot \infty = \infty \cdot 0 = 0$. For vectors $\mathbf{u} = (u_1, \ldots, u_c) \in \mathbb{N}^c_\infty$ and $\mathbf{v} = (v_1, \ldots, v_d) \in \mathbb{N}^d_\infty$, we denote by $\mathbf{u}\cdot\mathbf{v} = (u_1, \ldots, u_c, v_1, \ldots, v_d) \in \mathbb{N}^{c+d}_\infty$ the *concatenation of* $\mathbf{u}$ *and* $\mathbf{v}$. We extend this definition to sets of vectors. Let $C \subseteq \mathbb{N}^c_\infty$ and $D \subseteq \mathbb{N}^d_\infty$. Then $C \cdot D = \{\mathbf{u} \cdot \mathbf{v} \mid \mathbf{u} \in C, \mathbf{v} \in D\} \subseteq \mathbb{N}^{c+d}_\infty$. We denote by $\mathbf{0}^d$ the d-dimensional all-zero vector, by $\mathbf{1}^d$ the d-dimensional all-one-vector, by $\mathbf{e}^d_i$ the d-dimensional vector where the ith entry is 1 and all other entries are 0, and by $\mathbf{i}^d_i$ the d-dimensional vector where the ith entry is ∞ and all other entries are 0. We often drop the superscript d if the dimension is clear. For all complexity theoretical results, we assume a binary and explicit encoding of semi-linear sets unless stated otherwise.

[1] In contrast to the intuition given above, the transitions in our definition allow PDA to peek at a constant number of stack symbols at the same time. It is well-known that this generalization does not increase the expressiveness, as a PDA that may only peek at the top-most symbol can easily simulate this behavior using additional states and ε-transitions. Hence, we stick to the succinct definition as it is more convenient to work with.

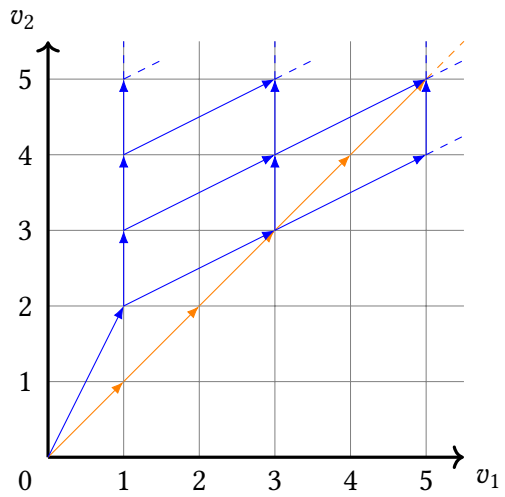


Figure 1.1. The semi-linear set $C = C((0,0),\{(1,1)\}) \cup C((1,2),\{(0,1),(2,1)\})$.

Semi-linear sets coincide with sets definable in Presburger arithmetic, that is, the first-order theory of (non-negative) integers and order, in the following sense: A Presburger formula $\varphi(\mathbf{v})$ with d free variables defines the set $\{\mathbf{v} \in \mathbb{N}^d \mid (\mathbb{N}, 0, 1, +, <) \models \varphi(\mathbf{v})\}$. By a groundbreaking result of Ginsburg and Spanier [GS64], a set $C \subseteq \mathbb{N}^d$ is semi-linear if and only if it is definable in Presburger arithmetic. Observe that this result implies that semi-linear sets are closed under the Boolean operations. We refrain from formally introducing first-order logic and refer to the textbooks [CJK13, Lib04] for an in-depth introduction. Instead, we outline the connection by the following example. Consider the linear sets $C_1 = C((0,0),\{(1,1)\})$ and $C_2 = C((1,2),\{(0,1),(2,1)\})$, and let $C = C_1 \cup C_2$, see Figure 1.1 for an illustration. We can define C_1 and C_2 by the Presburger formulas

$$\varphi_1(x_1, x_2) = \exists z.(x_1 = z \wedge x_2 = z)$$

and

$$\varphi_2(x_1, x_2) = \exists z_1.\exists z_2.(x_1 = 1 + 2z_2 \wedge x_2 = 2 + z_1 + z_2),$$

respectively (where 2 is an abbreviation for $1 + 1$ and $2z_2$ is an abbreviation for $z_2 + z_2$). Hence, for each period vector we guess the number of iterations using existential quantifiers. Consequently, we can define C by

$$\varphi(x_1, x_2) = \varphi_1(x_1, x_2) \vee \varphi_2(x_1, x_2).$$

The translation of Presburger formulas into semi-linear sets involves way more sophisticated methods, including a procedure to eliminate quantifiers; we refer the interested reader to [CH16, Haa18].

1.2.5 Parikh Recognizable Languages

A *Parikh automaton* (PA) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ where Q, Σ, q_0, and F are defined as for NFA, $\Delta \subseteq Q \times \Sigma \times \mathbb{N}^d \times Q$, for some $d \geq 1$, is a finite set of *labeled transitions*, and $C \subseteq \mathbb{N}^d$ is a semi-linear set. We call d the *dimension* of $\mathcal{A}$ and interpret d as a number of *counters*. Analogously to NFA, we call $\mathcal{A}$ *deterministic* if for every pair $(p, a) \in Q \times \Sigma$ there is exactly one labeled transition of the form $(p, a, \mathbf{v}, q) \in \Delta$ for some $\mathbf{v} \in \mathbb{N}^d$ and $q \in Q$. A *run* of $\mathcal{A}$ on a word $w = w_1 \dots w_n$ is a (possibly empty) sequence of labeled transitions $r = r_1 \dots r_n$ with $r_i = (p_{i-1}, w_i, \mathbf{v}_i, p_i) \in \Delta$ such that $p_0 = q_0$. We define the *extended Parikh image* of a run r as $\rho(r) = \sum_{i \leq n} \mathbf{v}_i$ (with the convention that the empty sum equals $\mathbf{0}$). We say r is accepting if $p_n \in F$ and $\rho(r) \in C$, referring to the latter condition as the *Parikh condition*. The *language recognized by* $\mathcal{A}$ is

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \text{there is an accepting run of } \mathcal{A} \text{ on } w\}.$$

If a language $L \subseteq \Sigma^*$ is recognized by some PA, then we call L *Parikh recognizable*.

Throughout the part we mainly deal with the Abelian group $(\mathbb{Z}^d, +, \mathbf{0})$ for some $d \geq 1$.

1.2.6 Directed Graphs

A *(directed) graph* G consists of its vertex set $V(G)$ and edge set $E(G) \subseteq V(G) \times V(G)$. In particular, a graph G may have loops, that is, edges of the form (u, u). A *(simple) path* from a vertex u to a vertex v in G is a sequence of pairwise distinct vertices $v_1 \dots v_k$ such that $v_1 = u$, $v_k = v$, and $(v_i, v_{i+1}) \in E(G)$ for all $1 \leq i < k$. Similarly, a *(simple) cycle* in G is a sequence of pairwise distinct vertices $v_1 \dots v_k$ such that $(v_i, v_{i+1}) \in E(G)$ for all $1 \leq i < k$, and $(v_k, v_1) \in E(G)$. If G has no cycles, we call G a directed acyclic graph (DAG). For a subset $U \subseteq V(G)$, we denote by $G[U]$ the graph G *induced by* U, i. e., the graph with vertex set U and edge set $\{(u, v) \in E(G) \mid u, v \in U\}$. A *strongly connected component* (SCC) in G is a maximal subset $U \subseteq V(G)$ such that for all $u, v \in U$ there is a path from u to v, i. e., all vertices in U are reachable from each other. We write $SCC(G)$ for the set of all strongly connected components of G (observe that $SCC(G)$ partitions $V(G)$). The *condensation* of G, written $C(G)$, is the DAG obtained from G by contracting each SCC of G into a single vertex, that is $V(C(G)) = SCC(G)$ and $(U, V) \in E(C(G))$ if and only if there is $u \in U$ and $v \in V$ with $(u, v) \in E(G)$. We call the SCCs with no outgoing edges in $C(G)$ leaves. Note that an automaton can be seen as a labeled graph. Hence, all definitions translate to automata by considering the underlying graph (to be precise, an automaton can be seen as a labeled multigraph; however, we simply drop parallel edges).

1.2.7 Turing Machines, Decidability and Complexity Theory

Turing machines are the most common model to define the notions of computability and decidability. Informally, a Turing machine is a finite automaton equipped with an infinite tape (or a constant number of such tapes) of discrete cells that can store symbols, and a read/write head (per tape) that points on one cell at a time. The Turing machine may read the symbol of the cell the head is pointing at, write a symbol in the cell, and move the head to a neighboring cell. However, when designing algorithms it is highly inconvenient to construct Turing machines implementing these algorithms. Instead, we will always give high level descriptions. For this reason, we refrain from formally defining Turing machines and refer the interested reader to the textbooks [HMU06, Koz97, Sip13]. If a language $L \subseteq \Sigma^*$ is recognized by a Turing machine, then we call L *semi-decidable*. Furthermore, if L is recognized by a Turing machine that halts on every input word, then we call L *decidable*. Similarly, if a function f is computed by a Turing machine, then we call f *computable*. Note that we can always represent a decision problem as a language. As decision problems often involve abstract objects as numbers, sets of numbers, automata, and so on, we always assume that such objects are encoded as words in an appropriate way. Hence, if a decision problem is decidable, we can implement an algorithm that solves it (in a finite amount of time). On the other hand, if a decision problem is undecidable, there is no algorithm solving it. Showing undecidability (and sometimes decidability) usually involves the notion of reductions. A *(many-one) reduction* from a problem $L \subseteq \Sigma^*$ to a problem $L' \subseteq \Gamma^*$ is a total, computable function $f : \Sigma^* \to \Gamma^*$ such that $w \in L$ if and only if $f(w) \in L'$. Hence, we can understand a reduction as an algorithm that translates instances of one problem to equivalent instances of a second problem. We write $L \leq L'$ if there is a reduction from L to L'. The following lemma obviously holds.

Lemma 1.2.2. *Let $L \subseteq \Sigma^*$ and $L' \subseteq \Gamma^*$ be two languages with $L \leq L'$.*

- *If L' is decidable, then L is decidable.*
- *If L is undecidable, then L' is undecidable.*

Very often we are not just interested in the question whether a problem is decidable, but in its inherent complexity, that is, the resources needed to solve it. We denote by P the complexity class of decision problems decidable in polynomial time by a deterministic Turing machine. Similarly, we denote by NP the complexity class of decision problems that are decidable in polynomial time by a nondeterministic Turing machine. Obviously, $\mathsf{P} \subseteq \mathsf{NP}$ holds. The question whether this inclusion is strict is the most famous question in theoretical computer science. While it is widely believed that the inclusion is strict, this problem is open for more than 50 years [Coo71]. The complexity class coNP contains the complements of problems in NP, that is, $\mathsf{coNP} = \{\overline{L} \mid L \in \mathsf{NP}\}$. The question whether NP and coNP are equivalent is also open.

A *polynomial time reduction* is a reduction computable in polynomial time by a deterministic Turing machine. We write $L \leq_{\mathsf{P}} L'$ if there is a polynomial time reduction from L to L'. We call a language L *hard*[2] for a complexity class C if $L' \leq_{\mathsf{P}} L$ for all $L' \in C$. If L is C-hard and contained in C, we call L *C-complete.* Intuitively, the notion of C-hardness yields a lower bound for the inherent complexity of a problem, while containment in C yields an upper bound. Hence, the notion of C-completeness yields a tight bound on the inherent complexity of a problem. Similar to the previous lemma, we will make constantly use of the following.

Lemma 1.2.3. *Let C be a complexity class and $L \subseteq \Sigma^*$ and $L' \subseteq \Gamma^*$ be two languages with $L \leq_{\mathsf{P}} L'$.*

- *If L' is contained in C, then L is contained in C.*
- *If L is hard for C, then L' is hard for C.*

An *oracle machine* with oracle $L \subseteq \Sigma^*$ is a Turing machine with a designated oracle tape. Such a machine can write a word w on its oracle type and answer the question $w \in L$ in one step. Let L be a language. Similar to P and NP we define the complexity classes P^L and NP^L of decision problems decidable in polynomial time by a deterministic resp. nondeterministic oracle machine with oracle L. Likewise, the complexity class coNP^L contains the complements of problems in NP^L. We extend these notion to classes: let $\mathcal{L}$ be a complexity class. Then we define

$$\mathsf{P}^{\mathcal{L}} = \bigcup_{L \in \mathcal{L}} \mathsf{P}^L; \qquad \mathsf{NP}^{\mathcal{L}} = \bigcup_{L \in \mathcal{L}} \mathsf{NP}^L; \qquad \mathsf{coNP}^{\mathcal{L}} = \bigcup_{L \in \mathcal{L}} \mathsf{coNP}^L.$$

Finally, we define the polynomial hierarchy. Let $\Delta_0^{\mathsf{P}} = \Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$. For every $i > 0$ we define

$$\Delta_i^{\mathsf{P}} = \mathsf{P}^{\Sigma_{i-1}^{\mathsf{P}}}, \qquad \Sigma_i^{\mathsf{P}} = \mathsf{NP}^{\Sigma_{i-1}^{\mathsf{P}}}, \qquad \Pi_i^{\mathsf{P}} = \mathsf{coNP}^{\Sigma_{i-1}^{\mathsf{P}}}.$$

The polynomial hierarchy PH is the union of all these classes for all $i \geq 0$.

1.2.8 Grammars and the Chomsky Hierarchy

A *grammar* is a tuple $G = (N, \Sigma, P, S)$, where N is the alphabet of non-terminals, Σ with $N \cap \Sigma = \varnothing$ is the alphabet of terminals, $P \subseteq N^+ \times (N \cup \Sigma)^*$ is the set of production rules, and $S \in N$ is the start symbol. For pairs $(\eta, \zeta_1), \ldots (\eta, \zeta_n) \in P$ for some $n \geq 1$ we write

[2] Depending on the class C, the notion of polynomial time reduction must be strengthened or can be weakened. For example, when showing NLogSpace-hardness, we require logarithmic space reductions. However, the given definition of hardness holds for all complexity classes considered in this part and we will hence ignore this technicality.

$\eta \to \zeta_1 \mid \cdots \mid \zeta_n$ instead. Let $w, w' \in (N \cup \Sigma)^*$. We say w *derives* into w' in G, written $w \vdash_G w'$ if $w = x\eta y$, $w' = x\zeta y$ and $\eta \to \zeta \in P$. A *derivation sequence* in G is a sequence $w_1, \ldots, w_n$ for some $n \geq 1$ such that $w_i \vdash_G w_{i+1}$ for all $1 \leq i \leq n$. We write $w \vdash_G^* w'$ if there is a derivation sequence $w_1, \ldots, w_n$ in G with $w = w_1$ and $w' = w_n$. We define the language generated by G as $L(G) = \{w \in \Sigma^* \mid S \vdash_G^* w\}$. Observe that $L(G)$ contains only words over Σ.

We say a grammar $G = (N, \Sigma, P, S)$ is of ...

- *type-0* if there are no restrictions on the rules in P.
- *type-1* if for every rule $\eta \to \zeta \in P$ we have $|\eta| \leq |\zeta|$. Furthermore, the rule $S \to \varepsilon$ is allowed if S does not appear on the right handside of any rule in P.
- *type-2* if every rule in P is of the form $A \to \eta$ with $A \in N$.
- *type-3* if every rule in P is of the form $A \to wB$ or $A \to w$ with $A, B \in N$ and $w \in \Sigma^*$.

We denote the class of language generated by type-i grammars by $\mathcal{L}_i$. The famous Chomsky-hierarchy, named after the linguist Noam Chomsky, states

$$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0.$$

Hence, $\mathcal{L}_0$ is the class of languages captured by grammars in their full generality, while restricting the production rules yield less expressive grammars. It is well-known that all these classes can equivalently be defined in terms of automata, as clarified by the following folklore lemma.

Lemma 1.2.4. *Let L be a language. The following statements hold.*

- *$L \in \mathcal{L}_0$ if and only if L is semi-decidable.*
- *$L \in \mathcal{L}_1$ if and only if L is decidable by a Turing machine whose tapes are bounded by the input length.*
- *$L \in \mathcal{L}_2$ if and only if L is context-free.*
- *$L \in \mathcal{L}_3$ if and only if L is regular.*

2 Parikh Automata on Finite Words

2.1 Parikh Automata and Friends

In this chapter we show some results for Parikh automata on finite words. While these results are interesting on their own, some of them yield algorithms that we will use as subroutines in later sections.

Klaedtke and Ruess [KR03b] showed that the class of Parikh recognizable languages is equivalent to the class of languages recognized by Ibarra's reversal bounded multicounter machines [Iba78], the class of language recognized by weighted automata over the group $(\mathbb{Z}^d, +, \mathbf{0})$ as introduced by Mitrana and Stiebe [MS01], which is by definition equivalent to the class of languages of Greibach's blind multicounter machines [Gre78], which again is equivalent to the class of reachability languages of integer vector addition systems with states ($\mathbb{Z}$-VASS) [HH14]. The latter models are very similar to Parikh automata, the difference being that transitions are labeled with vectors of (possible negative) integers, and a run is accepting if an accepting state is reached at the end while all counters are 0. Baumann et al. showed that all these models can be translated into each other using only logarithmic space [BDG+23]. However, the number of counters is in general not preserved in these translations.

Let us formally define blind multicounter machines at this point, as considering such machines (instead of PA) simplifies some proofs in this section. A *blind multicounter machine* (BMCM) is a tuple $\mathcal{M} = (Q, \Sigma, q_0, \Delta, F)$, where Q, Σ, q_0 and F are defined as for NFA, and $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{Z}^d \times Q$ for some $d \geq 1$ is a finite set of transitions labeled with vectors of (possibly negative) integers. Note that ε-transitions are allowed. As for PA, we call d the dimension of $\mathcal{M}$ and interpret d as a number of counters. A *configuration* of $\mathcal{M}$ is a tuple $(q, w, \mathbf{v}) \in Q \times \Sigma^* \times \mathbb{Z}^d$, where q is the current state, w is a suffix of the input word that has not been read yet, and $\mathbf{v}$ represents the current counter values. Let c, c' be two configurations of $\mathcal{M}$. We say c *derives* into c', written $c \vdash_{\mathcal{M}} c'$, if $c = (q, aw, \mathbf{v})$, $c' = (q', w, \mathbf{v}')$ and $(q, a, \mathbf{u}, q') \in \Delta$ with $\mathbf{v}' = \mathbf{v} + \mathbf{u}$; or if $c = (q, w, \mathbf{v})$, $c' = (q', w, \mathbf{v}')$ and $(q, \varepsilon, \mathbf{u}, q') \in \Delta$ with $\mathbf{v}' = \mathbf{v} + \mathbf{u}$. A *computation* of $\mathcal{M}$ is a sequence of configurations $c_1, \ldots, c_n$ for some $n \geq 1$ with $c_i \vdash_{\mathcal{M}} c_{i+1}$ for all $1 \leq i < n$. We write $c \vdash^*_{\mathcal{M}} c'$ if there is a computation $c_1, \ldots, c_n$ with $c = c_1$ and $c_n = c'$. We define the langugae recognized by $\mathcal{M}$ as $L(\mathcal{M}) = \{w \in \Sigma^* \mid (q_0, w, \mathbf{0}) \vdash (q, \varepsilon, \mathbf{0}) \text{ for a } q \in F\}$. Latteux [Lat79] has shown that BMCM of dimension d admit an effective procedure to eliminate ε-transitions.

Lemma 2.1.1 ([Lat79]). *For every BMCM of dimension d with ε-transitions there is an equivalent BMCM of dimension d with no ε-transitions.*

As mentioned above, although PA and BMCM are known to be equivalent, the translations do in general not preserve the dimension. We first show that these models are equivalent up to one dimension. This result will come in handy in this section, as it allows us to translate fine-grained results from one model to the other. First, we observe that we can in a sense consider the linear sets of the union of the semi-linear set of a PA independently.

Observation 2.1.2. *Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ with $C = \bigcup_{i \leq \ell} C(\mathbf{b}_i, P_i)$ be a PA. Then $L(\mathcal{A}) = \bigcup_{i \leq \ell} L(Q, \Sigma, q_0, \Delta, F, C(\mathbf{b}_i, P_i))$.*

Lemma 2.1.3. *The following are equivalent for every language L.*

(1) L is recognized by a BMCM of dimension $d \geq 1$.

(2) L is recognized by a PA of dimension $d + 1$.

Proof. The implication (1) $\Rightarrow$ (2) is very similar to the construction in [JK03, Theorem 4.5]; we present a proof sketch. Let $\mathcal{M} = (Q, \Sigma, q_0, \Delta, F)$ be a BMCM of dimension d and for $(v_1, \ldots, v_d) \in \mathbb{Z}^d$ define $\|(v_1, \ldots, v_d)\| = \max\{|v_1|, \ldots, |v_d|\}$. Let

$$D = \max\{\|\mathbf{v}\| \mid (p, a, \mathbf{v}, q) \in \Delta \text{ for some } p, q \in Q \text{ and } a \in \Sigma\}$$

be the largest absolute value appearing in a transition of $\mathcal{M}$. We construct a PA $\mathcal{A}$ of dimension $d+1$ with the same state space as $\mathcal{M}$ as follows. For every transition $(p, a, (v_1, \ldots, v_d), q)$ in Δ, we insert the transition $(p, a, (v_1 + D, \ldots, v_d + D, D), q)$ in $\mathcal{A}$. Note that every entry in $(v_1 + D, \ldots, v_d + D, D)$ is non-negative by the choice of D. In particular, whenever a counter of $\mathcal{M}$ is of value v, then the same counter of $\mathcal{A}$ is of value $v - v_D$, where v_D denotes the counter value of the newly introduced counter. In particular, when all counters of $\mathcal{M}$ are 0, then all counters of $\mathcal{A}$ have the same value. Hence, by choosing the linear set $C(\mathbf{0}, \{\mathbf{1}\})$, we obtain an equivalent PA $\mathcal{A}$ with only one additional counter as desired.

For the implication (2) $\Rightarrow$ (1), let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a PA of dimension $d + 1$. By the previous observation we may assume that $C = C(\mathbf{b}, P)$ is a linear set, as we can carry out the following procedure translating $\mathcal{A}$ into an equivalent BMCM $\mathcal{M}$ for every linear set independently and combine resulting machines by introducing a fresh initial state with outgoing ε-transitions to the initial states of the $\mathcal{M}$, which can then be eliminated by Lemma 2.1.1 without changing the dimension. We construct $\mathcal{M}$ from $\mathcal{A}$ as follows. First, we interpret $\mathcal{A}$ as a BMCM, that is, we keep the states and transitions of $\mathcal{A}$ and forget the linear set. Next, for every $\mathbf{p} \in P$, we equip every state of $\mathcal{A}$ with an ε-transition subtracting $\mathbf{p}$. Furthermore, we introduce a new accepting state, say q_f, connect every $q \in F$ to q_f via an ε-transition subtracting $\mathbf{b}$, and make every other state non-accepting, that is, we set $F = \{q_f\}$. Let $\mathcal{M}$ be the BMCM obtained this way. First observe that these modifications do not change the accepted language of $\mathcal{A}$, that is $L(\mathcal{A}) = L(\mathcal{M})$. Let m be the largest integer appearing in the first component of a vector in P. Now observe that for every run of $\mathcal{A}$ there is a run of $\mathcal{M}$ such that the first counter value stays in the set $[0, m]$. Indeed, whenever the first counter value would exceed m, we can first subtract a fitting period vector using one of the introduced ε-loops. This is always possible, as the counter values might be negative in $\mathcal{M}$. Hence, we may assume that the first counter value stays within a fixed range, and hence we can simulate the first counter in the state space of $\mathcal{M}$, i. e., the state space of $\mathcal{M}$ is $Q \times [0, m]$, where Q is the state space of $\mathcal{A}$. Hence, we lose one dimension as desired. ◀

2.2 A Pumping Lemma for Parikh Recognizable Languages

The famous pumping lemma for regular languages essentially states that for every regular language L and every sufficiently long word w in L, we can partition w in three parts, say xyz such that we can pump (and depump) y and still obtain words in L, as stated in the following lemma.

Lemma 2.2.1. *Let L be a regular language.*
Then there exists a constant $p \in \mathbb{N}$
such that for all $w \in L$ with $|w| \geq p$
there is a partition $w = xyz$ with $|xy| \leq p$ and $|y| > 0$
such that for all $k \in \mathbb{N}$ we have $xy^k z \in L$.

Observe that the pumping lemma only yields a necessary condition for a language being regular, not a sufficient one. This means, there are non-regular languages that satisfy the condition of the pumping lemma, e. g. the language

$$\{a^m b^n c^n \mid n, m \geq 1\} \cup \{b^m c^n \mid n, m \geq 0\}.$$

Jaffe established a variant of the pumping lemma that is both, necessary and sufficient [Jaf78].

A small modification of the pumping lemma for regular languages yields the well-known pumping lemma for context-free languages.

Lemma 2.2.2. *Let L be a context-free language.*
Then there exists a constant $p \in \mathbb{N}$
such that for all $w \in L$ with $|w| \geq p$
there is a partition $w = uvxyz$ with $|vxy| \leq p$ and $|vy| > 0$
such that for all $k \in \mathbb{N}$ we have $uv^k xy^k z \in L$.

For more information on the pumping lemmas, we refer the interested reader to the textbooks [HMU06, Koz97, Sip13].

In a similar spirit, Cadilhac et al. established the following "pumping-style" lemma yielding a necessary condition for a language being Parikh recognizable [CFM11].

Lemma 2.2.3. *Let L be a Parikh recognizable language.*
Then there exist constants $p, \ell \in \mathbb{N}$
such that for all $w \in L$ with $|w| \geq \ell$
there is a partition $w = uvxvz$ with $0 < |v| < p < |x|$ and $uvxv < \ell$
such that $uv^2xz \in L$ and $uxv^2z \in L$.

Note that, unlike the pumping lemma for regular languages, the pumping-style lemma states that sufficiently long words w in a Parikh recognizable languages can be split in

five parts $uvxvz$ such that a short infix v of w appears twice in w and its position can be switched.

In this section we prove the following pumping lemma yielding a necessary condition for languages being recognizable by PA of dimension d.

Lemma 2.2.4. *Let L be a language recognized by a PA of dimension d.*
Then there exists a constant $p \in \mathbb{N}$
such that for all $w \in L$ with $|w| \geq p$
there is a partition $w = u_0 v_1 u_1 \dots v_d u_d$ with $|v_1 \dots v_d| > 0$
and a vector $(x_1, \dots x_d) \in \mathbb{N}^d$
such that for all $k \in \mathbb{N}$ we have $u_0 v_1^{1+kx_1} u_1 \dots v_d^{1+kx_d} u_d \in L$.

Intuitively, the lemma states that the number of infixes we may pump independently matches the dimension d. This can in general be not avoided, as witnessed by the following class of languages. Define for every $k \geq 1$ the alphabet $\Sigma_k = \{a_1, \dots, a_k\}$ and let $L_k = \{a_1^n \dots a_k^n \mid n \geq 0\}$. Obviously, L_k is recognized by a PA with k counters, and we for every $w \in L_k$ we need to pump k infixes independently, namely one block of a_i for every $1 \leq i \leq k$. Furthermore observe that this pumping lemma almost generalizes the pumping lemmas for regular and context-free languages in the sense that the special cases for $d = 1$ and $d = 2$ yield only slightly weaker statements than Lemma 2.2.1 and Lemma 2.2.2. This matches with the results in Lemma 2.3.1 and Lemma 2.3.2 presented in the next section.

Before we prove the lemma, we need the following notion. Let $\leq$ be a binary relation on a set X. We call $\leq$ a *quasi-order* (on X) if it satisfies the following properties.

Reflexivity. For all $x \in X$ we have $x \leq x$.

Transitivity. For all $x, y, z \in X$ we have $x \leq y$ and $y \leq z$ implies $x \leq z$.

We call a quasi-order $\leq$ on a set X a *well-quasi-order* if it neither contains infinite descending chains nor infinite anti-chains, that is, if every infinite sequence $x_1, x_2, \dots$ of pairwise distinct elements of X contains two elements x_i, x_j with $i < j$ and $x_i \leq x_j$. Let $Y \subseteq X$ and $x \in Y$. We call x *minimal* with respect to $\leq$ if Y contains no smaller element, that is, there is no element $y \neq x$ in Y with $y \leq x$. Observe that for every well-quasi-order $\leq$ on X, every subset $Y \subseteq X$ has only a finite number of minimal elements with respect to $\leq$. Higman's famous lemma [Hig52] states that the subword relation $\preccurlyeq$ on Σ^* is a well-quasi-order. We are now ready to prove Lemma 2.2.4. In the light of Lemma 2.1.3 it is sufficient to show the following pumping lemma for BMCM.

Lemma 2.2.5. *Let L be a language recognized by a BMCM of dimension d.*
Then there exists a constant $p \in \mathbb{N}$
such that for all $w \in L$ with $|w| \geq p$
there is a partition $w = u_0 v_1 u_1 \dots v_{d+1} u_{d+1}$ with $|v_1 \dots v_{d+1}| > 0$
and a vector $(x_1, \dots x_{d+1}) \in \mathbb{N}^d$
such that for all $k \in \mathbb{N}$ we have $u_0 v_1^{1+kx_1} u_1 \dots v_{d+1}^{1+kx_{d+1}} u_{d+1} \in L$.

Proof. Let $\mathcal{M} = (Q, \Sigma, q_0, \Delta, F)$ be a BMCM of dimension d recognizing L. By Lemma 2.1.1 we may assume that $\mathcal{M}$ has no ε-transitions. Let $C = c_1, \ldots, c_n$ be a computation of $\mathcal{M}$ with $c_i = (q_i, w_i, \mathbf{v}_i)$ for all $1 \leq i \leq n$. We associate with C the run $r(C) = r_1 \ldots r_{n-1} \subseteq \Delta^*$, that is, the sequence of transitions witnessing $c_i \vdash_{\mathcal{M}} c_{i+1}$ for all $1 \leq i < n$. Let $r = r(C)$. Borrowing the notation from PA, we denote by $\rho(r) = \mathbf{v}_n - \mathbf{v}_1$, which is equivalent to the sum of the vectors appearing in the r_i. As every run can be seen as word over Δ, we apply Higman's lemma and obtain that the subword relation $\preccurlyeq$ over Δ is a well-quasi-order and hence the subword relation over the set of accepting runs. This implies that there is a finite set of minimal subruns, say $r^{(1)}, \ldots, r^{(m)}$ with $\rho(r^{(i)}) = \mathbf{0}$ for all $i \leq m$. Let p be the maximal length among these subruns. Let $w \in L$ with $|w| \geq p$, and let C be a computation witnessing membership of w in L. As the run $r = r(C)$ must have some $r^{(i)} = r_0^{(i)} r_1^{(i)} \ldots r_\ell^{(i)}$ as a subrun, we can write $r = r_0^{(i)} \tau_1 r_1^{(i)} \ldots \tau_\ell r_\ell^{(i)}$ with $\tau_i \in \Delta^*$ for all $1 \leq i \leq \ell$. Now observe that every τ_i is a cycle in $\mathcal{M}$. Furthermore, we have $\rho(\tau_1) + \cdots + \rho(\tau_\ell) = \mathbf{0}$, as $\rho(r^{(i)}) = \rho(r) = \mathbf{0}$. By [FS02, Lemma 6.2] there is a subset $\{\tau'_1, \ldots \tau'_k\} \subseteq \{\tau_1, \ldots, \tau_\ell\}$ with $k \leq d + 1$ such that there are $x_1, \ldots, x_k \in \mathbb{N}$ with $x_1\rho(\tau'_1) + \cdots + x_k\rho(\tau'_k) = \mathbf{0}$. Let $v_1, \ldots v_k$ be the infixes processed in $\tau'_1, \ldots, \tau'_k$, respectively, and let $\tau_{k+1}, \ldots, \tau_{d+1}$ be the empty words. Hence, we obtain the desired partition of w by filling the u_i with the remaining infixes. ◀

This lemma allows a more fine-grained analysis of Parikh recognizable languages. Take as an example the language $\{a^n b^n \mid n \geq 0\}$, which is obviously Parikh recognizable with two counters, and also context-free. Note however that, as a consequence of the pumping lemma, there is no PA with only one counter recognizing this language. Similarly, the language $\{a^n b^n c^n \mid n \geq 0\}$ is Parikh recognizable with three counters but there is no PA with only two recognizing it. Furthermore, this language is not context-free. These observations motivate the next section, where we compare the classes of languages recognized by PA with $d \geq 1$ counters with the classes in the Chomsky hierarchy.

2.3 Parikh Automata and the Chomsky Hierarchy

In this section we show that the class of languages recognized by PA with a single counter coincides with $\mathcal{L}_3$, while the class of languages recognized by PA with two counters is a strict subset of $\mathcal{L}_2$. Furthermore, we translate known hierarchy results for BMCM to PA. We complete the picture by a result in [CFM12a] stating that every Parikh recognizable language is contained in $\mathcal{L}_1$. We refer to Figure 2.1 for an overview.

Lemma 2.3.1. *The following are equivalent for every language L.*

(1) L is regular.

(2) L is recognized by a PA with a single counter.

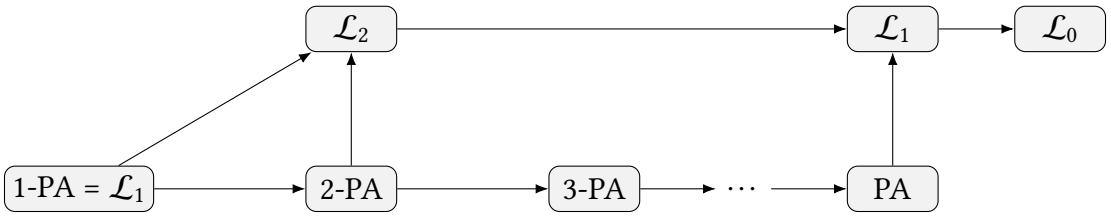


Figure 2.1. Location of Parikh recognizable languages in the Chomsky hierarchy. The term d-PA refers to the class of languages recognized by PA of dimension d. Arrows indicate strict inclusions while no (non-transitive) connections mean incomparability.

Proof. The implication (1) $\Rightarrow$ (2) is clear, as we can always turn an NFA into an equivalent PA of dimension 1 with linear set $\{0\}$ by labeling every transition with 0.

For the implication (1) $\Rightarrow$ (2) let $\mathcal{A} = (Q, \Sigma, q_0, \Sigma, F, C)$ be a PA of dimension 1. By Observation 2.1.2 we may assume that C is linear as the class of regular languages is closed under union; hence, we can carry our the following procedure for each linear set individually. First, we show how to translate $\mathcal{A}$ into an equivalent NFA $\mathcal{A}'$ if C has a very simple form, namely that $C = (\mathbf{b}, \{\mathbf{p}\})$, that is, a linear set with a single (one-dimensional) period vector. Now we simulate the counter of $\mathcal{A}$ in the state space of $\mathcal{A}'$ as follows: as long as the counter value does not exceed $\mathbf{b}$, we store the current counter value in the state space of $\mathcal{A}$. As soon as the counter exceeds this threshold, we use a fresh copy of the state space of $\mathcal{A}'$ to store the "distance" to the next multiple of $\mathbf{p}$, that is, the current counter value modulo $\mathbf{p}$. Formally, let $\mathcal{A}' = (Q', \Sigma, (q_0, 0), \delta', F')$ be an NFA with

$$Q' = \{(q, i) \mid q \in Q, 0 \leq i < \mathbf{b}\} \cup \{(q', i) \mid q' \in Q, 0 \leq i < \mathbf{p}\},$$
$$F' = \{(q', 0) \mid q' \in F\},$$

where the set of transitions is defined as follows:

$$\begin{aligned}\Delta' = &\{((p, i), a, (q, i + j)) \mid (p, a, j, q) \in \Delta, i + j < \mathbf{b}\} \\ &\cup \{((p, i), a, (q', (i + j - \mathbf{b}) \mod \mathbf{p})) \mid (p, a, j, q) \in \Delta, i + j \geq \mathbf{b}\} \\ &\cup \{((p', i), a, (q', (i + j) \mod \mathbf{p})) \mid (p, a, j, q) \in \Delta\}.\end{aligned}$$

In the next step, we assume that $C = (\mathbf{b}, P)$ is an arbitrary linear set. We use (a small modification of) Bézout's identity [Bé79], stating that for all non-negative integers m, n, and $k \geq mn$ that are multiples of $\gcd(m, n)$, there are non-negative integers c, d such that $k = mc + nd$, where $\gcd(m, n)$ denotes the greatest common divisor of m and n. We extend the definition of gcd to sets the natural way. Thus, we can equally describe C as a union of the finite set $C' = \{n \in C \mid n < \mathbf{b} + \Pi_{\mathbf{p} \in P}\mathbf{p}\}$ and a linear set with only one period vector, namely $C(\mathbf{b} + \Pi_{\mathbf{p} \in P}\mathbf{p}, \{\gcd(P)\})$. Now we can use a construction very similar to the

previous case, encoding all values of the finite set as well the new period vector into the state space of $\mathcal{A}'$, additionally making all states (q, i) with $i \in C'$ accepting. Note that $\mathcal{A}'$ has $|Q|(\mathbf{b} + \Pi_{\mathbf{p}\in P}\mathbf{p} + \gcd(P))$-many states. ◀

Lemma 2.3.2. *Let L be a language recognized by a PA with two counters. Then L is context-free.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a PA of dimension 2. Again, we assume that $C = ((b_1, b_2), P)$ is linear as the class of context-free languages is closed under union. In the first step, we build a type-3 grammar G_C that generates words over the alphabet $\{x, y\}$, where x, y are fresh symbols that do not appear in Σ, with the property that a vector $\mathbf{v}$ is member of C if and only if there is a word $w \in L(G_C)$ with $\mathbf{v} = (|w|_x, |w|_y)$. Hence, let $G_C = (\{S\}, \{x, y\}, P_C, S)$ with

$$P_C = \{S \to x^{p_1}y^{p_2}S \mid (p_1, p_2) \in P\} \cup \{S \to x^{b_1}y^{b_2}\}.$$

In the next step, we modify G_C in such a way that we obtain a context-free grammar G'_C with the property that $\mathbf{v} \in C$ if and only if $L(G'_C)$ contains *all* words w with $(|w|_x, |w|_y) \in C$, and no other words. Particularly, G'_C generates all words that are permutations of words generated by G'_C. It is known that the set of permutations of words of a regular language over an alphabet with two symbols is context-free (see e. g. [Sin09]). Hence, let G'_C be a context-free grammar generating the permutations of all words in $L(G_C)$.

By a standard construction, we can build a PDA $\mathcal{M}_C$ with a single state that accepts exactly the words generated by G'_C by simulating a derivation sequence where we always derive the leftmost non-terminal first (using ε-transitions) on the stack and checking the input word against the derived word on the stack (see e. g. [Koz97] for details). Hence, let $\mathcal{M}_C$ be such a PDA with $L(\mathcal{M}_C) = L(G'_C)$.

Finally, we build a PDA from $\mathcal{A}$ inheriting the stack alphabet and ε-transition from $\mathcal{M}_C$. The idea is as follows. Let r be an accepting run of $\mathcal{A}$ with $\rho(r) = \mathbf{v} \in C$. Recall that $\mathcal{M}_C$ accepts all words w with $\mathbf{v} = (|w|_x, |w|_y)$. Now, for every transition $(p, a, (v_1, v_2), q) \in \Delta$ we expect the word $x^{v_1}y^{p_1}$ to be on top of the stack. That is, we construct $\mathcal{M}$ as follows. We begin with a copy of $\mathcal{A}$ and replace every transition $(p, a, (v_1, v_2), q)$ with $(p, a, x^{v_1}y^{v_2}, \varepsilon, q)$. At the beginning, we put a fresh stack symbol, say #, at the bottom of the stack. Furthermore, we copy the ε-transitions in $\mathcal{M}_C$ (simulating the derivation sequences of G'_C) to every state of $\mathcal{A}_C$. Finally, we may only remove the new symbol # in the accepting states of $\mathcal{A}$. This finishes the construction.

We sketch that we indeed have $L(\mathcal{A}) = L(\mathcal{M})$. Let $w \in L(\mathcal{A})$ with accepting run $r = r_1 \dots r_n$ collecting the vector $\mathbf{v} = (v_1 + \dots + v_n, v'_1 + \dots + v'_n) \in C$. As $\mathcal{M}_C$ accepts the word $x^{v_1}y^{v'_1} \dots x^{v_n}y^{v'_n}$, the PDA $\mathcal{M}$ can always prepare the stack such that the transition $(p, a, x^{v_i}y^{v'_i}, \varepsilon, q)$ is eligible whenever the PA $\mathcal{A}$ takes the corresponding transition. Finally,

as r ends in an accepting state, $\mathcal{M}$ may safely remove the new stack symbol #, emptying the stack.

Now let $w \in L(\mathcal{M})$ and let $u \in \{x, y\}^*$ be the word simulated on the stack while processing w. In particular, we have $u \in L(\mathcal{M}_C)$ and hence $(u_{|x|}, u_{|y|}) \in C$. Furthermore, as w is in an accepting state of $\mathcal{A}$ after processing w, we conclude $w \in L(\mathcal{A})$. ◀

Remark. Klaedtke and Ruess claim in [KR03b, example in Definition 3] that there is a 2-PA for the language $\{a^{i+j}b^ic^j \mid i, j \geq 0\}$, and that this language is not context-free. However, this language is context-free, so there is no contradiction.

We note that the inclusion is strict, as the language

$$\mathsf{Pal} = \{w_1 \dots w_n \in \{a, b\}^* \mid n \geq 0, w_1 \dots w_n = w_n \dots w_1\}$$

of palindromes is context-free but not recognized by any PA. This can be shown using the pumping-style lemma (Lemma 2.2.3) very similar to the proof of [CFM11, Proposition 3] showing that the language $\mathsf{Copy} = \{w\#w \mid w \in \{a, b\}^*\}$ is not Parikh recognizable. In fact, we choose the word $(a^pb)^\ell(ba^p)^\ell \in \mathsf{Pal}$ (instead of $(a^pb)^\ell\#(a^pb)^\ell \in \mathsf{Copy}$) and re-use the arguments in their proof.

The following lemma is an immediate consequence from [CFM12a, Proposition 4.12].

Lemma 2.3.3. *Let L be a Parikh recognizable language. Then $L \in \mathcal{L}_1$.*

The results in [Lat79] imply that BMCM with $d \geq 1$ counters are strictly less expressive than BMCM with $d + 1$ counters. This results translates immediately to PA by Lemma 2.1.3.

Corollary 2.3.4. *For every $d \geq 1$, there is a language L that is a language that is recognized by no PA of dimension d but by a PA of dimension $d + 1$.*

Furthermore, Latteux [Lat79] has shown that every language L recognized by a blind counter machine can be written as a finite intersection of languages recognized by blind counter machines of dimension 1. Hence, we obtain the following.

Corollary 2.3.5. *Let L be a Parikh recognizable language. Then there are languages $L_1, \dots, L_n$ for some $n \geq 1$ recognized by PA of dimension 2 such that $L = \bigcap_{i \leq n} L_i$.*

Note that the dimension in this corollary is tight in the sense that PA of dimension 1 recognize regular languages only by Lemma 2.3.1, which are closed under intersection. Furthermore, the corollary implies that for every $d \geq 2$ the class of languages recognized by PA of dimension d is not closed under intersection.

2.4 Universality for Deterministic Parikh Automata

In this section, we consider the following common decision problems for (deterministic) PA.

- Emptiness: given a PA $\mathcal{A}$, is $L(\mathcal{A}) = \emptyset$?
- Membership: given a PA $\mathcal{A}$ and a finite words w, is $w \in L(\mathcal{A})$?
- Universality: given a PA $\mathcal{A}$, is $L(\mathcal{A}) = \Sigma^*$?

As shown by Klaedtke and Ruess [KR03b], emptiness and membership are decidable for deterministic and non-deterministic PA. Furthermore, they have shown that universality is decidable for deterministic PA but undecidable for non-deterministic PA. Refining these results, Figueira and Libkin [FL15] showed that non-emptiness and membership are NP-complete for non-deterministic PA, and these results translate quickly to the deterministic setting as well.

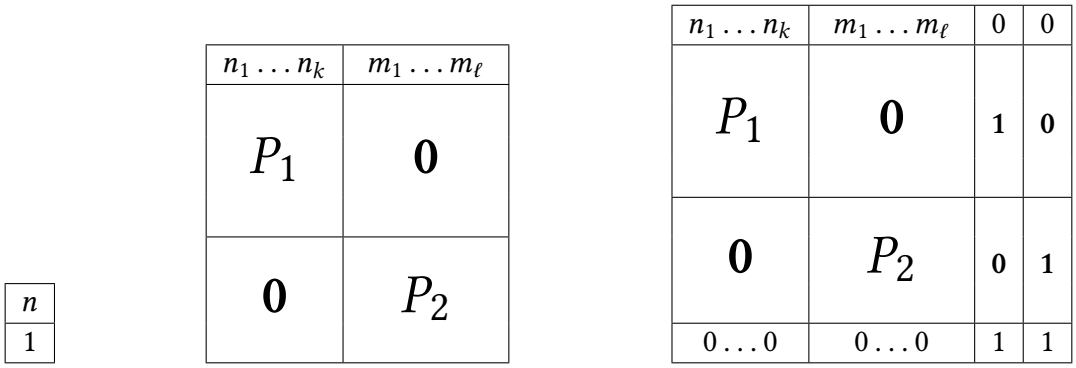
As a preparation for the results in Section 3.2 we show that universality for deterministic PA is Π_2^{P}-complete in order to establish complexity bounds of several problems related to PA operating on infinite words.

To achieve that, we first introduce an auxiliary problem for PA and show that it is already Π_2^{P}-hard even restricted to deterministic acyclic PA. We define the *irrelevance problem* for PA as follows: given a PA $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ of dimension d, is $\mathcal{A}$ equivalent to $\mathcal{A}^* = (Q, \Sigma, q_0, \Delta, F, \mathbb{N}^d)$? In other words: is C irrelevant for $\mathcal{A}$ in the sense that every run of $\mathcal{A}$ reaching an accepting state satisfies the Parikh condition? In the following, we denote by $\rho(\mathcal{A}) = \{\rho(r) \mid r \text{ is an accepting run of } \mathcal{A}^*\}$ and note that this set is always semi-linear as a consequence of Parikh's theorem [Par66] and [KR03b, Lemma 5].

Lemma 2.4.1. *The irrelevance problem for deterministic acyclic PA is Π_2^{P}-hard.*

Proof. We reduce from the inclusion problem for integer expressions, which is known to be Π_2^{P}-complete (assuming that all numbers are encoded in binary) [SM73, Sto76]. The set of integer expressions is defined as follows. Every $n \in \mathbb{N}$ is an integer expression with $L(n) = \{n\}$. If e_1 and e_2 are integer expressions, then so are $e_1 + e_2$ and $e_1 \cup e_2$ where $L(e_1 + e_2) = \{u + v \mid u \in L(e_1), v \in L(e_2)\}$ and $L(e_1 \cup e_2) = L(e_1) \cup L(e_2)$. The inclusion problem is defined as follows: given integer expressions e_1, e_2, is $L(e_1) \subseteq L(e_2)$?

We proceed as follows: first, we construct a linear set $C(\mathbf{0}, P_2)$ of dimension $d + 1$ from e_2 with the property that $n \in L(e_2)$ if and only if $n \cdot \mathbf{1}^d \in C(\mathbf{0}, P_2)$, where d and $|P_2|$ depend linearly on the number of operators in e_2. Second, we construct a deterministic acyclic PA $\mathcal{A}_1$ from e_1 with the property $L(e_1) = \rho(\mathcal{A}_1)$. Finally, we construct a deterministic acyclic PA $\mathcal{A}$ from $\mathcal{A}_1$ such that $C(\mathbf{0}, P_2)$ is irrelevant for $\mathcal{A}$ if and only if $L(e_1) \subseteq L(e_2)$.



(a) $e = n$ (b) $e = e_1 + e_2$ (c) $e = e_1 \cup e_2$

Figure 2.2. Illustration of the construction of a linear set from an integer expression.

Claim 2.4.2. *Given an integer expression e, we can compute in polynomial time a linear set $C(\mathbf{0}, P)$ such that $n \in L(e)$ if and only if $n \cdot \mathbf{1} \in C(\mathbf{0}, P)$.*

Proof. We construct $C(\mathbf{0}, P)$ inductively from e and maintain the following invariant in each step: $n \in L(e)$ if and only if $n \cdot \mathbf{1} \in C(\mathbf{0}, P)$ and for all $\mathbf{p} = (m, p_1 \dots, p_d) \in P$ we have $p_i = 1$ for at least one $1 \leq i \leq d$. We refer to Figure 2.2 for an illustration.

Base Case. If $e = n$ for $n \in \mathbb{N}$, we choose $C(\mathbf{0}, P) \in \mathbb{N}^2$ with $P = \{(n, 1)\}$.

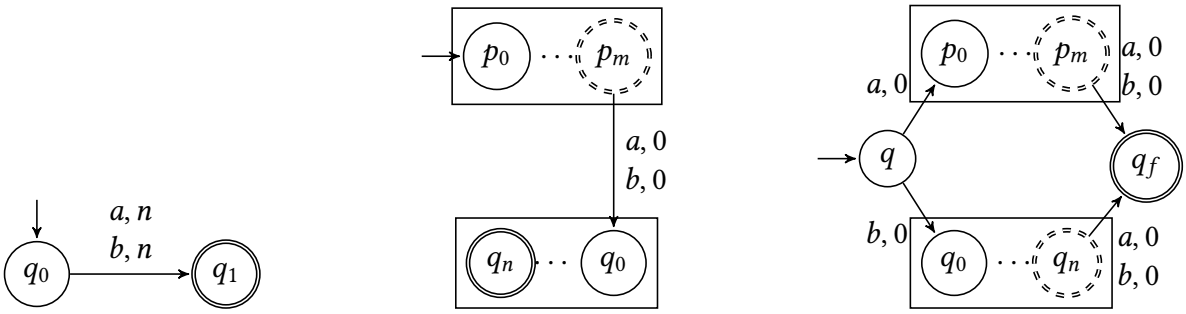
Step. We need to consider two cases.

If $e = e_1 + e_2$, there are linear sets $C(\mathbf{0}, P_1) \subseteq \mathbb{N}^{1+d_1}$ and $C(\mathbf{0}, P_2) + \mathbb{N}^{1+d_2}$ for e_1 and e_2 satisfying the invariant by assumption. We construct a linear set $C(\mathbf{0}, P)$ of dimension $1 + d_1 + d_2$ as follows. We pad the vectors in P_1 and P_2 with zeros to align the dimensions. Then P is the union of these vectors, i. e.,

$$P = \{n \cdot \mathbf{p}_1 \cdot \mathbf{0}^{d_2} \mid n \cdot \mathbf{p}_1 \in P_1\} \cup \{n \cdot \mathbf{0}^{d_1} \cdot \mathbf{p}_2 \mid n \cdot \mathbf{p}_2 \in P_2\}.$$

The invariant is maintained using this construction: for every integer $m + n \in L(e)$ we have $m \cdot \mathbf{1}^{d_1} \cdot \mathbf{0}^{d_2} \in C(\mathbf{0}, \{m \cdot \mathbf{p}_1 \cdot \mathbf{0}^{d_2} \mid m \cdot \mathbf{p}_1 \in P_1\})$ and $n \cdot \mathbf{0}^{d_1} \cdot \mathbf{1}^{d_2} \in C(\mathbf{0}, \{n \cdot \mathbf{0}^{d_2} \cdot \mathbf{p}_2 \mid n \cdot \mathbf{p}_2 \in P_2\})$ by assumption and construction. Hence, $(n+m) \cdot \mathbf{1}^{d_1+d_2} \in C(\mathbf{0}, P)$. Likewise, if $n \cdot \mathbf{1}^{d_1+d_2} \in C(\mathbf{0}, P)$, we can partition the sets of used period vectors in the set of period vectors originating from P_1, and the set originating from P_2. As only the period vectors originating from P_1 can modify the first d_1 counters (ignoring the first), they yield a number $n_1 \in L(e_1)$. Similarly, the period vectors from p_2 yield a number $n_2 \in L(e_2)$, and hence $n = n_1 + n_2 \in L(e)$.

If $e = e_1 \cup e_2$, there are linear sets $C(\mathbf{0}, P_1) \subseteq \mathbb{N}^{1+d_1}$ and $C(\mathbf{0}, P_2) \subseteq \mathbb{N}^{1+d_2}$ for e_1 and e_2 satisfying the invariant by assumption. We construct a linear set $C(\mathbf{0}, P)$ of dimension $1 + d_1 + d_2 + 1$ as as follows. Again, we pad the vectors in P_1 and P_2 with zeros to align the dimensions, and add an additional 0-counter. Furthermore, we consider the two vectors



(a) $e = n$ (b) $e = e_1 + e_2$ (c) $e = e_1 \cup e_2$

Figure 2.3. Illustration of the construction of a deterministic acyclic PA from an integer expression.

$\mathbf{v}_1 = 0 \cdot \mathbf{1}^{d_1} \cdot \mathbf{0}^{d_2} \cdot 1$ and $\mathbf{v}_2 = 0 \cdot \mathbf{0}^{d_1} \cdot \mathbf{1}^{d_2} \cdot 1$. Then P is the union of these vectors, i. e.,

$$P = \{n \cdot \mathbf{p}_1 \cdot \mathbf{0}^{d_2} \cdot 0 \mid n \cdot \mathbf{p}_1 \in P_1\} \cup \{n \cdot \mathbf{0}^{d_1} \cdot \mathbf{p}_2 \cdot 0 \mid n \cdot \mathbf{p}_2 \in P_2\} \cup \{\mathbf{v}_1, \mathbf{v}_2\}.$$

The invariant is maintained using this construction: if $n \in L(e_1)$, then $n \cdot \mathbf{1}^{d_1+d_2+1} \in C(\mathbf{0}, P)$ as witnessed by the following choice of period vectors. First, we can use $\mathbf{v}_2$ to ensure that the last $d_2 + 1$ entries (including the new counter) are indeed set to one. Furthermore, the first $d_1 + 1$ entries of $\mathbf{v}_1$ (including the first counter) are 0; hence, they do not modify the relevant counters for n. Hence, the containment follows from the invariant and construction. We argue analogously if $n \in L(e_2)$ using $\mathbf{v}_1$. Now, if $n \cdot \mathbf{1}^{d_1+d_2+1} \in C(\mathbf{0}, P)$ we observe that exactly one of the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ must have been used, as these are the only ones that modify the last counter. If $\mathbf{v}_1$ has been used, then no period vector originating in P_1 may have been used, as they all contain a one-entry by the invariant, and are hence blocked by the 1-entries in $\mathbf{v}_1$. As $\mathbf{v}_1$ does not modify the first counter, and all used period vectors originate from P_2, we conclude $n \in L(e_2)$. Analogously, if $\mathbf{v}_2$ has been used, we conclude $n \in L(e_1)$.

Observe that the dimension d and the size of P both depend linearly on the size of e, hence $|C(\mathbf{0}, P)| \in O(|e|^2)$. ◁

Claim 2.4.3. *Given an integer expression e, we can compute in polynomial time a deterministic acyclic PA with a single accepting state $\mathcal{A}$ such that $L(e) = \rho(\mathcal{A})$.*

Proof. We construct $\mathcal{A}$ over the alphabet $\{a, b\}$ of dimension 1 with linear set $\mathbb{N}$ inductively from e and maintain the invariant in the claim in every step. We refer to Figure 2.3 for an illustration.

Base Case. If $e = n$ for $n \in \mathbb{N}$, the PA $\mathcal{A}$ consists of an initial state and an accepting state that are connected by an a-transition and a b-transition, both labeled with n.

Step. We need to consider two cases.

If $e = e_1 + e_2$, there are PA $\mathcal{A}_1$ and $\mathcal{A}_2$ for e_1 and e_2 satisfying the invariant by assumption. We construct a PA $\mathcal{A}$ as follows: we take the disjoint union of $\mathcal{A}_1$ and $\mathcal{A}_2$ and connect the accepting state of $\mathcal{A}_1$ with the initial state of $\mathcal{A}_2$ via an a-transition and a b-transitions, both labeled with 0. Finally, the accepting states of $\mathcal{A}_1$ is not accepting anymore in $\mathcal{A}$.

If $e = e_1 \cup e_2$, there are PA $\mathcal{A}_1$ and $\mathcal{A}_2$ for e_1 and e_2 satisfying the invariant by assumption. We construct a PA $\mathcal{A}$ as follows: we take the disjoint union of $\mathcal{A}_1$ and $\mathcal{A}_2$ and add a fresh initial state, say q, and a fresh accepting state, say q_f. Then, we connect q with the initial state of $\mathcal{A}_1$ with an a-transition labeled with $\mathbf{0}$, and we connect q with the initial state of $\mathcal{A}_2$ with a b-transition labeled with 0. Similarly, we connect the accepting state of $\mathcal{A}_1$ as well as the accepting state of $\mathcal{A}_2$ with q_f via a-transitions and b-transitions, all labeled with 0. Finally, the accepting states of $\mathcal{A}_1$ and $\mathcal{A}_2$ are not accepting anymore in $\mathcal{A}$. ◁

We are now ready to prove Lemma 2.4.1. Let $\mathcal{A}_1$ be the PA for e_1 as constructed in the proof of Claim 2.4.3. Similarly, let $C(\mathbf{0}, P_2)$ be the linear set of dimension $1 + d$ for e_2 as constructed in the proof of Claim 2.4.2. Now we construct the deterministic acyclic PA $\mathcal{A}$ as follows. We start with $\mathcal{A}_1$ and pad all vectors with zeros, that is, we replace every (one-dimensional) vector n in $\mathcal{A}_1$ by $n \cdot \mathbf{0}^d$. Then, we add a fresh accepting state and connect it from the accepting state of $\mathcal{A}_1$ with an a-transition and b-transition, both labeled with $0 \cdot \mathbf{1}^d$. Finally, the accepting state of $\mathcal{A}_1$ is not accepting anymore in $\mathcal{A}$, and we choose $C(\mathbf{0}, P_2)$ as the linear set of $\mathcal{A}$. Observe that the properties of $\mathcal{A}_1$ and $C(\mathbf{0}, P_2)$ ensure that $C(\mathbf{0}, P_2)$ is irrelevant for $\mathcal{A}$ if and only if $L(e_1) \subseteq L(e_2)$. This concludes the proof. ◀

Remark. Recall that we assume a binary and explicit encoding of semi-linear sets. In terms of expressiveness we can equally assume that they are given as Presburger formulas. However, this drastically changes the complexity: if the semi-linear sets are given as Presburger formulas, the problem becomes coNEXP-complete, as we can interreduce the problem with the $\forall^*\exists^*$-fragment of Presburger arithmetic, which is coNEXP-complete [Haa14].

Observe that we can easily reduce the previous problem to the universality problem for deterministic PA.

Corollary 2.4.4. *Universality for deterministic PA is Π_2^{P}-hard.*

Proof. Let $\mathcal{A}$ be the deterministic acyclic PA with linear set $C(\mathbf{0}, P)$ as constructed in the previous proof. We construct a deterministic $\mathcal{A}'$ such that $C(\mathbf{0}, P)$ is irrelevant for $\mathcal{A}$ if and only if $\mathcal{A}'$ is universal. To achieve that, we start with $\mathcal{A}$ and make every state accepting. Furthermore, we add an a-loop and a b-loop to the accepting state of $\mathcal{A}$, both labeled with $\mathbf{0}$. Finally, the semi-linear set of $\mathcal{A}'$ is $C(\mathbf{0}, P) \cup (\mathbb{N} \cdot \{\mathbf{0}\})$. ◀

Now we show that universality for deterministic PA is in Π_2^{P}, yielding completeness for universality and irrelevance by the reduction presented in the proof of the previous corollary.

Lemma 2.4.5. *Universality for deterministic PA is Π_2^{P}-complete.*

Proof. We show that the problem is contained in Π_2^{P} by showing that its complement is contained in Σ_2^{P}. We make use of the following observation. As we can assume that every state is accepting (by encoding accepting states into the semi-linear set), the question whether a deterministic PA $\mathcal{A}$ with semi-linear set C is not universal boils down to the question $\rho(\mathcal{A}) \not\subseteq C$? If both sets are given explicitly (again assuming binary encoding), Huynh showed that this question can be decided in Σ_2^{P} [Huy86, Huy80]. However, we cannot explicitly compute the set $\rho(\mathcal{A})$ as its size can be exponentially large in the size of $\mathcal{A}$ [To10]. To circumvent this problem, we guess a small linear subset of $\rho(\mathcal{A})$ that witnesses non-inclusion.

In the first step, we compute an existential Presburger formula, say $\varphi(\mathbf{v}) = \exists x_1 \dots x_m \psi$ defining the set $\rho(\mathcal{A})$. This can be done in linear time using the results in [VSS05]; we also refer to [FL15, Proposition III.2] for a short discussion on the construction.

In the second step, we use the result in [HZ21, Corollary II.2] essentially stating that every semi-linear set can be written as a finite union of linear sets of small bit size. To be precise, the result states that every semi-linear set $C \subseteq \mathbb{N}^d$ can be written as $\bigcup_i C(\mathbf{b}_i, P_i)$ with $|P_i| \leq 2d \log(4d\|C\|)$, where $\|C\|$ denotes the largest absolute value that appears in a base or period vector in any linear set in the union of C. Hence, we guess a linear set $C(\mathbf{b}_i, P_i) \subseteq \rho(\mathcal{A})$ of small bit size. Note however, that we do not verify at this point whether $C(\mathbf{b}_i, P_i)$ is indeed a subset of $\rho(\mathcal{A})$.

In the third step, we use Huynh's results [Huy86, Huy80] to solve $C(\mathbf{b}_i, P_i) \not\subseteq C$ in Σ_2^{P}. As one main ingredient to obtain containment Σ_2^{P}, Huynh showed that every non-empty (set-theoretic) difference of two semi-linear sets contains a vector of polynomial bit size. Hence let $\mathbf{v} \in C(\mathbf{b}_i, P_i) \setminus C$ be such a vector whose bit size is bounded polynomially in $C(\mathbf{b}_i, P_i)$ and C, and hence in the input size.

Finally, we need to verify that $\mathbf{v}$ is indeed a member of $\rho(\mathcal{A})$ (recall that we did not verify that $C(\mathbf{b}_i, P_i)$ is indeed a subset of $C_{\mathcal{A}}$). In order to do so, we guess a valuation of the quantified variables $x_1, \dots, x_n$ (again of polynomial bit size) of φ and verify that $\varphi(\mathbf{v})$ is indeed satisfied under this valuation.

Overall, we conclude that non-universality for deterministic PA is in Σ_2^{P}, yielding the desired result. ◀

3 Parikh Automata on Infinite Words

In this chapter, we recall the acceptance conditions of Parikh automata operating on infinite words that were studied before in the literature and introduce our new models. We make some easy observations and compare the existing with the new automata models. First we focus on the non-deterministic variants of these models before studying their deterministic variants. Whenever we do not explicitly specify that a model is deterministic, we mean the non-deterministic variant.

Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a PA. A run of $\mathcal{A}$ on an infinite word $\alpha = \alpha_1\alpha_2\alpha_3\ldots$ is an infinite sequence of labeled transitions $r = r_1r_2r_3\ldots$ with $r_i = (p_{i-1}, \alpha_i, \mathbf{v}_i, p_i) \in \Delta$ such that $p_0 = q_0$. The automata defined below differ only in their acceptance conditions; hence, the notion of determinsm translates directly. In the following, whenever we say that an automaton $\mathcal{A}$ accepts an infinite word α, we mean that there is an accepting run of $\mathcal{A}$ on α.

1. The run r satisfies the *safety condition* if for every $i \geq 0$ we have $p_i \in F$ and $\rho(r_1 \ldots r_i) \in C$. We call a PA accepting with the safety condition a *safety PA* [GJLZ22]. We define the ω-language recognized by a safety PA $\mathcal{A}$ as

$$S_\omega(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}.$$

2. The run r satisfies the *reachability condition* if there is an $i \geq 1$ such that $p_i \in F$ and $\rho(r_1 \ldots r_i) \in C$. We say there is an *accepting hit* in r_i. We call a PA accepting with the reachability condition a *reachability PA* [GJLZ22]. We define the ω-language recognized by a reachability PA $\mathcal{A}$ as

$$R_\omega(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}.$$

3. The run r satisfies the *Büchi condition* if there are infinitely many $i \geq 1$ such that $p_i \in F$ and $\rho(r_1 \ldots r_i) \in C$. We call a PA accepting with the Büchi condition a *Büchi PA* [GJLZ22]. We define the ω-language recognized by a Büchi PA $\mathcal{A}$ as

$$B_\omega(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}.$$

   Hence, a Büchi PA can be seen as a stronger variant of a reachability PA where we require infinitely many accepting hits instead of a single one.

4. The run r satisfies the *co-Büchi condition* if there is i_0 such that for every $i \geq i_0$ we have $p_i \in F$ and $\rho(r_1 \ldots r_i) \in C$. We call a PA accepting with the co-Büchi condition a *co-Büchi PA* [GJLZ22]. We define the ω-language recognized by a co-Büchi PA $\mathcal{A}$ as

$$CB_\omega(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}.$$

   Hence, a co-Büchi PA can be seen as a weaker variant of safety PA where the safety condition needs not necessarily be fulfilled from the beginning, but from some point onwards.

We now define the models newly introduced in this work. As already observed in [GJLZ22] among the above considered models only Büchi PA can recognize all ω-regular languages. For example, $\{\alpha \in \{a,b\}^\omega \mid |\alpha|_a = \infty\}$ cannot be recognized by safety PA, reachability PA or co-Büchi PA.

We first extend reachability PA with the classical Büchi condition to obtain *reachability-regular PA*. In Theorem 3.1.8 we show that these automata characterize the class

$$\mathcal{L}^{\omega}_{\mathsf{PA},\mathsf{Reg}} = \left\{ \bigcup_{i \le n} U_i V_i \mid n \ge 1, U_i \text{ is Parikh-recognizable}, V_i \text{ is regular} \right\}$$

hence, providing a robust and natural model.

5. The run satisfies the *reachability and regularity condition* if there is an $i \ge 1$ such that $p_i \in F$ and $\rho(r_1 \dots r_i) \in C$, and there are infinitely many $j \ge 1$ such that $p_j \in F$. We call a PA accepting with the reachability and regularity condition a *reachability-regular* PA. We define the ω-language recognized by a reachability-regular PA $\mathcal{A}$ as

$$RR_\omega(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}.$$

Observe that every ω-regular language is reachability-regular PA recognizable, as we can turn an arbitrary Büchi automaton into an equivalent reachability-regular PA by labeling every transition with 0 and setting $C = \{0\}$.

Next we introduce *limit PA*, which were proposed in the concluding remarks of [KR03b]. As we will prove in Theorem 3.1.8, this seemingly quite different model is equivalent to reachability-regular PA.

6. The run satisfies the *limit condition* if there are infinitely many $i \ge 1$ such that $p_i \in F$, and if additionally $\rho(r) \in C$, where the jth component of $\rho(r)$ is computed as follows. If there are infinitely many $i \ge 1$ such that the jth component of $\mathbf{v}_i$ has a non-zero value, then the jth component of $\rho(r)$ is ∞. In other words, if the sum of values in a component diverges, then its value is set to ∞. Otherwise, the infinite sum yields a positive integer. We call a PA accepting with the limit condition a *limit PA*. We define the ω-language recognized by a limit PA $\mathcal{A}$ as

$$L_\omega(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}.$$

Still, none of the yet introduced models have ω-closure. This shortcoming is addressed with the following two models, which will turn out to be equivalent and form the basis of the automata characterization of the classes

$$\mathcal{L}^{\omega}_{\mathsf{PA},\mathsf{PA}} = \left\{ \bigcup_{i \le n} U_i V_i \mid n \ge 1, U_i \text{ and } V_i \text{ are Parikh-recognizable} \right\}$$

and

$$\mathcal{L}^{\omega}_{\text{Reg,PA}} = \left\{ \bigcup_{i \le n} U_i V_i \mid n \ge 1, U_i \text{ is regular, } V_i \text{ is Parikh-recognizable} \right\}.$$

7. The run satisfies the *strong reset condition* if the following holds. Let $k_0 = 0$ and denote by $k_1 < k_2 < \dots$ the positions of all accepting states in r. Then r is accepting if $k_1, k_2, \dots$ is an infinite sequence and $\rho(r_{k_{i-1}+1} \dots r_{k_i}) \in C$ for all $i \ge 1$. We call a PA accepting with the strong reset condition a *strong reset PA*. We define the ω-language recognized by a strong reset PA $\mathcal{A}$ as

$$SR_\omega(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}.$$

8. The run satisfies the *weak reset condition* if there are infinitely many reset positions $0 = k_0 < k_1 < k_2, \dots$ such that $p_{k_i} \in F$ and $\rho(r_{k_{i-1}+1} \dots r_{k_i}) \in C$ for all $i \ge 1$. We call a PA accepting with the weak reset condition a *weak reset PA*. We define the ω-language recognized by a weak reset PA $\mathcal{A}$ as

$$WR_\omega(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}.$$

Intuitively worded, whenever a strong reset PA enters an accepting state, the Parikh condition *must* be satisfied. Then the counters are reset. Similarly, a weak reset PA may reset the counters whenever there is an accepting hit, and they must reset infinitely often, too. In the following we will often just speak of reset PA without explicitly stating whether they are weak or strong. In this case, we mean the strong variant. We will show the equivalence of the nondeterministic variants of two models in Lemma 3.1.23 and Lemma 3.1.24, while in the deterministic setting weak reset PA are more expressive than strong reset PA, see Lemma 3.2.17.

Guha et al. [GJLZ22] assume that reachability PA are complete, i.e., for every $(p, a) \in Q \times \Sigma$ there are $\mathbf{v} \in \mathbb{N}^d$ and $q \in Q$ such that $(p, a, \mathbf{v}, q) \in \Delta$, as incompleteness allows to express additional safety conditions. We also make this assumption in order to avoid inconsistencies. In fact, we can assume that all models are complete, as the other models can be completed by adding a non-accepting sink. Observe that their deterministic variants are always complete by definition.

Example 3.0.1. Let $\mathcal{A}$ be the automaton in Figure 3.1 with $C = \{(z, z'), (z, \infty) \mid z' \ge z\}$.

- If we interpret $\mathcal{A}$ as a PA (over finite words), then we have $L(\mathcal{A}) = \{w \in \{a, b\}^* \cdot \{b\} \mid |w|_a \le |w|_b\} \cup \{\varepsilon\}$. The automaton is in the accepting state at the very beginning and every time after reading a b. The first counter counts the occurrences of a, the second one counts occurrences of b. By definition of C the automaton only accepts when the second counter value is greater or equal to the first counter value (note that vectors containing an ∞-entry have no additional effect).

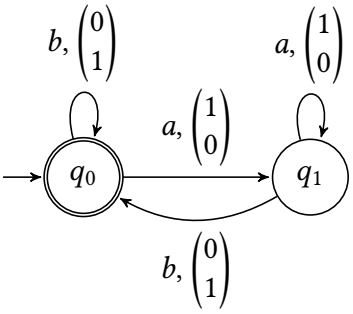


Figure 3.1. The automaton $\mathcal{A}$ with $C = \{(z, z'), (z, \infty) \mid z' \geq z\}$ from Example 3.0.1.

- If we interpret $\mathcal{A}$ as a safety PA, then we have $S_\omega(\mathcal{A}) = \{b\}^\omega$. As q_1 is not accepting, only the b-loop on q_0 may be used.
- If we interpret $\mathcal{A}$ as a reachability PA, then we have $R_\omega(\mathcal{A}) = \{\alpha \in \{a, b\}^\omega \mid \alpha \text{ has a prefix in } L(\mathcal{A})\}$. The automaton has satisfied the reachability condition after reading a prefix in $L(\mathcal{A})$ and accepts any continuation after that.
- If we interpret $\mathcal{A}$ as a Büchi PA, then we have $B_\omega(\mathcal{A}) = L(\mathcal{A})^\omega$. The automaton accepts an infinite word if infinitely often the Parikh condition is satisfied in the accepting state. Observe that C is a homogeneous linear set and the initial state as well as the accepting state have the same outgoing transitions.
- If we interpret $\mathcal{A}$ as a co-Büchi PA, then we have $CB_\omega(\mathcal{A}) = L(\mathcal{A}) \cdot \{b\}^\omega$. This is similar to the safety PA, but the accepted words may have a finite "non-safe" prefix from $L(\mathcal{A})$.
- If we interpret $\mathcal{A}$ as a reachability-regular PA, then we have $RR_\omega(\mathcal{A}) = \{\alpha \in \{a, b\}^\omega \mid \alpha \text{ has a prefix in } L(\mathcal{A}) \text{ and } |\alpha|_b = \infty\}$. After having met the reachability condition the automaton still needs visit an accepting state infinitely often.
- If we interpret $\mathcal{A}$ as a limit PA, then we have $L_\omega(\mathcal{A}) = \{\alpha \in \{a, b\}^\omega \mid |\alpha|_a < \infty\}$. The automaton must visit the accepting state infinitely often. At the same time the extended Parikh image must belong to C, which implies that the infinite word contains only some finite number z of symbol a (note that only the vectors of the form (z, ∞) have an effect here, as at least one symbol must be seen infinitely often by the infinite pigeonhole principle).
- If we interpret $\mathcal{A}$ as a weak reset PA, then we have $WR_\omega(\mathcal{A}) = L(\mathcal{A})^\omega$. As a weak reset PA may (but is not forced to) reset the counters upon visiting the accepting state, the automaton may reset every time a (finite) infix in $L(\mathcal{A})$ has been read.
- If we interpret $\mathcal{A}$ as a strong reset PA, then we have $SR_\omega(\mathcal{A}) = \{b^*a\}^\omega \cup \{b^*a\}^* \cdot \{b\}^\omega$. Whenever the automaton reaches an accepting state also the Parikh condition must be satisfied. This implies that the a-loop on q_1 may never be used, as this would increase the first counter value to at least 2, while the second counter value is 1 upon reaching the accepting state q_0 (which resets the counters).

3.1 Nondeterministic Parikh Automata

In this section, we study the nondeterministic variants of Parikh automata on infinite words. We first show that the newly introduced variants are closed under union and left-concatenation with Parikh recognizable languages, yielding the foundation in order to establish the claimed characterizations. These characterizations in turn help us to establish the remaining closure properties. Motivated by further examining the expressiveness of all these models, we study the effect of ε-transitions and show that almost all considered models admit ε-elimination, the exception being safety and co-Büchi PA. Based on this result, we show that Büchi PA and blind counter automata operating on infinite words as introduced by Fernau and Stiebe [FS08] are equivalent. Finally, we study the classical decision problems with application to model checking.

3.1.1 Preparation

It was observed in [GJLZ22] that Büchi PA recognize a strict subset of $\mathcal{L}^{\omega}_{\mathrm{PA,PA}}$. In this section we first show that the class of reset PA recognizable ω-languages is a strict superset of $\mathcal{L}^{\omega}_{\mathrm{PA,PA}}$. Then we provide an automata-based characterization of $\mathcal{L}^{\omega}_{\mathrm{PA,Reg}}$, $\mathcal{L}^{\omega}_{\mathrm{PA,PA}}$, and $\mathcal{L}^{\omega}_{\mathrm{Reg,PA}}$. Towards this goal we first establish some closure properties.

Guha et al. [GJLZ22] have shown that safety, reachability, Büchi, and co-Büchi PA are closed under union using a modification of the standard construction for PA, i. e., taking the disjoint union of the automata (introducing a fresh initial state), and the disjoint union of the semi-linear sets, where disjointness is achieved by "marking" every vector in the first set by an additional 1 (increasing the dimension by 1), and all vectors in the second set by an additional 2. Then all transitions from the new initial state leading to the copy of the first automaton are also marked with a 1 and those leading to the copy of the second automaton are marked with a 2. We observe that the same construction also works for reachability-regular and limit PA, and a small modification is sufficient to make the construction also work for reset PA. To be precise, we need to refresh the mark every time we leave an accepting state, as the reset "forgets" this information. We leave the details to the reader.

Lemma 3.1.1. *The classes of reachability-regular PA recognizable, limit PA recognizable, and reset PA recognizable ω-languages are closed under union.*

Furthermore, we show that these classes, as well as the class of Büchi PA recognizable ω-languages, are closed under left-concatenation with PA recognizable languages. We provide some details in the next lemma, as we will need to modify the standard construction in such a way that we do not need to keep accepting states of the PA on finite words. This will help to characterize $\mathcal{L}^{\omega}_{\mathrm{PA,PA}}$ via (restricted) reset PA.

Lemma 3.1.2. *The classes of reachability-regular PA recognizable, limit PA recognizable, reset PA recognizable, Büchi PA recognizable, co-Büchi PA recognizable, and safety PA recognizable ω-languages are closed under left-concatenation with PA recognizable languages.*

Proof. We begin with reset PA. Let $\mathcal{A}_1 = (Q_1, \Sigma, q_1, \Delta_1, F_1, C_1)$ be a PA of dimension d_1 and let $\mathcal{A}_2 = (Q_2, \Sigma, q_2, \Delta_2, F_2, C_2)$ be a reset PA of dimension d_2. We sketch the construction of a reset PA $\mathcal{A}$ of dimension $d_1 + d_2$ that recognizes $L(\mathcal{A}_1) \cdot SR_\omega(\mathcal{A})$. We assume without loss of generality that q_2 is accepting (this can be achieved by introducing a fresh initial state). Furthermore, for now we assume that $\varepsilon \notin L(\mathcal{A}_1)$, that is, every accepting run of $\mathcal{A}_1$ is not empty. Again, $\mathcal{A}$ consists of disjoint copies of $\mathcal{A}_1$ and $\mathcal{A}_2$ but only the accepting states of $\mathcal{A}_2$ remain accepting, and the initial state of $\mathcal{A}$ is q_1. All transitions of the copy of $\mathcal{A}_1$ use the first d_1 counters (that is, the remaining d_2 counters are always 0), and, likewise, the transitions of $\mathcal{A}_2$ only use the last d_2 counters (that is, the first d_1 counters are always 0). Finally, we copy every transition of $\mathcal{A}_1$ that leads to an accepting state of $\mathcal{A}_1$ such that it also leads to q_2, that is, we add the transitions $\{(p, a, \mathbf{v} \cdot 0^{d_2}, q_2) \mid (p, a, \mathbf{v}, q) \in \Delta_1, q \in F_1\}$. The semi-linear set C of $\mathcal{A}$ is $C_1 \cdot \{0^{d_2}\} \cup \{0^{d_1}\} \cdot C_2$. As every accepting run of $\mathcal{A}_1$ is non-empty by assumption, $\mathcal{A}$ may guess the last transition of every accepting run of $\mathcal{A}_1$ and replace it with one of the new transitions that leads to $\mathcal{A}_2$ instead. As q_2 is accepting, the counters are reset, which justifies the choice of C. Now, if $\varepsilon \in L(\mathcal{A}_1)$, observe that $L(\mathcal{A}_1) \cdot SR_\omega(\mathcal{A}_2) = (L(\mathcal{A}_1) \setminus \{\varepsilon\} \cdot SR_\omega(\mathcal{A}_2)) \cup SR_\omega(\mathcal{A}_2)$. Hence, we may remove ε from $L(\mathcal{A}_1)$ by replacing q_1 by a fresh non-accepting copy and use the closure under union. Hence, in any case only the copies of accepting states of $\mathcal{A}_2$ remain accepting; in particular no state of $\mathcal{A}_1$ is accepting in the corresponding copy of $\mathcal{A}$.

The construction for reachability-regular PA, limit PA and Büchi PA is very similar. The only difference is that we choose $C = C_1 \cdot C_2$ for the semi-linear set of $\mathcal{A}$, as counters are never reset here.

Finally, for co-Büchi PA and safety PA we need the following modifications. First, we make every state of $\mathcal{A}$ accepting. Furthermore, we add one additional counter that is set to 1 on every transition from an accepting state of $\mathcal{A}_1$ to the initial state of $\mathcal{A}_2$ (note that every run of $\mathcal{A}$ can only take such a transition once). This counter is not modified on every other transition. The new semi-linear set is $C = (\mathbb{N}^{d_1+d_2} \cdot \{0\}) \cup (C_1 \cdot C_2 \cdot \{1\})$, that is, as long as we have not reached the state set of $\mathcal{A}_2$, we do not check the counter values yet. ◀

Before we continue, we show that we can normalize PA (on finite words) such that the initial state is the only accepting state. This observation simplifies several proofs in this section.

Lemma 3.1.3. *Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a PA of dimension d. Then there exists an equivalent PA $\mathcal{A}'$ of dimension $d+1$ with the following properties.*

- *The initial state of $\mathcal{A}'$ is the only accepting state.*
- *$SCC(\mathcal{A}') = \{Q\}$.*

We say that $\mathcal{A}'$ is normalized.

Proof. The normalized PA $\mathcal{A}'$ is obtained from $\mathcal{A}$ by adding a fresh state q_0', which is the initial state and only accepting state, and which inherits all outgoing transitions from q_0 and all incoming transitions from the accepting states. Furthermore, all transitions get a new counter, which is set to 0 except for the new incoming transitions of q_0' where the counter is set to 1, and all vectors in C are concatenated with 1 (and we add the all zero-vector if we want to accept ε). Finally, we remove all states that cannot reach q_0' (such states can appear when shortcutting the incoming transitions of F, and are useless in the sense that their removal does not change the accepted language; however, this removal is necessary for the second property).

Formally, we define $\mathcal{A}' = \{Q \cup \{q_0'\}, \Sigma, q_0', \Delta', \{q_0'\}, C')$, where

$$\begin{aligned}\Delta' = &\{(p, a, \mathbf{v} \cdot 0, q) \mid (p, a, \mathbf{v}, q) \in \Delta\} \\ &\cup \{(q_0', a, \mathbf{v} \cdot 0, q) \mid (q_0, a, \mathbf{v}, q) \in \Delta\} \\ &\cup \{(p, a, \mathbf{v} \cdot 1, q_0') \mid (p, a, \mathbf{v}, f) \in \Delta, f \in F\} \\ &\cup \{(q_0', a, \mathbf{v} \cdot 1, q_0') \mid (q_0, a, \mathbf{v}, f) \in \Delta, f \in F\}.\end{aligned}$$

and

$$C' = \begin{cases} C \cdot \{1\} & \text{if } \varepsilon \notin L(\mathcal{A}) \\ C \cdot \{1\} \cup \{\mathbf{0}^{d+1}\} & \text{otherwise.} \end{cases}$$

It is now easily verified that $L(\mathcal{A}) = L(\mathcal{A}')$. ◀

Observe that we have $SR_\omega(\mathcal{A}') = L(\mathcal{A})^\omega$, that is, every normalized PA interpreted as a reset PA recognizes the ω-closure of the language recognized by the PA. As an immediate consequence we obtain the following corollary.

Corollary 3.1.4. *For every Parikh recognizable language L we have that L^ω is reset PA recognizable.*

Combining these results we obtain that every ω-language in $\mathcal{L}^\omega_{\mathsf{PA},\mathsf{PA}}$, i.e., every ω-language of the form $\bigcup_i U_i V_i^\omega$ is reset PA recognizable. We show that the other direction does not hold, i.e., the inclusion is strict.

Lemma 3.1.5. *The class $\mathcal{L}^{\omega}_{\mathrm{PA,PA}}$ is a strict subclass of the class of reset PA recognizable ω-languages.*

Proof. The inclusion is a direct consequence of Lemma 3.1.1, Lemma 3.1.2, and Corollary 3.1.4. Hence it remains to show that the inclusion is strict.

Consider the ω-language $L = \{a^n b^n \mid n \geq 1\}^{\omega} \cup \{a^n b^n \mid n \geq 1\}^* \cdot \{a\}^{\omega}$. This ω-language is reset PA recognizable, as witnessed by the strong reset PA in Figure 3.2 with semi-linear set $C = \{(z, z) \mid z \in \mathbb{N}\}$.

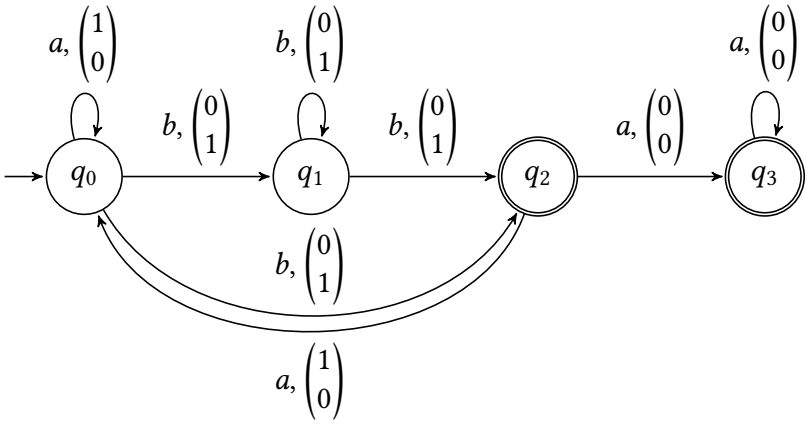


Figure 3.2. The strong reset PA for $L = \{a^n b^n \mid n \geq 1\}^{\omega} \cup \{a^n b^n \mid n \geq 1\}^* \cdot \{a\}^{\omega}$ with semi-linear set $C = \{(z, z) \mid z \in \mathbb{N}\}$.

We claim that $L \notin \mathcal{L}^{\omega}_{\mathrm{PA,PA}}$. Assume towards a contraction that $L \in \mathcal{L}^{\omega}_{\mathrm{PA,PA}}$, i. e., there are Parikh recognizable languages $U_1, V_1, \ldots, U_n, V_n$ such that $L = U_1 V_1^{\omega} \cup \cdots \cup U_n V_n^{\omega}$. Then there is some $i \leq n$ such that for infinitely many $j \geq 1$ we have $\alpha_j = aba^2b^2 \ldots a^j b^j \cdot a^{\omega} \in U_i V_i^{\omega}$. Then V_i must contain a word of the form $v = a^k$, $k > 0$. Additionally, there cannot be a word in V_i with infix b. To see this assume for sake of contradiction that there is a word $w \in V_i$ with $\ell = |w|_b > 0$. Let $\beta = (v^{\ell+1} w)^{\omega}$. Observe that β has an infix that consists of at least $\ell + 1$ many a, followed by at most ℓ, but at least one b, hence, no word of the form $u\beta$ with $u \in U_i$ is in L. This is a contradiction, thus $V_i \subseteq \{a\}^+$.

Since U_i is Parikh recognizable, there is a PA $\mathcal{A}_i$ with $L(\mathcal{A}_i) = U_i$. Let m be the number of states in $\mathcal{A}_i$ and $w' = aba^2b^2 \ldots a^{m^4+1}b^{m^4+1}$. Then w' is a prefix of a word accepted by $\mathcal{A}_i$. Now consider the infixes $a^{\ell}b^{\ell}$ and the pairs of states q_1, q_2, where we start reading a^{ℓ} and end reading a^{ℓ}, and q_3, q_4 where we start to read b^{ℓ} and end to read b^{ℓ}, respectively. There are m^2 choices for the first pair and m^2 choices for the second pair, hence m^4 possibilities in total. Hence, as we have more than m^4 such infixes, there must be two with the same associated states q_1, q_2, q_3, q_4. Then we can swap these two infixes and get a word of the form $ab \ldots a^r b^s \ldots a^s b^r \ldots a^{m^4+1} b^{m^4+1}$ that is a prefix of some word in $L(\mathcal{A}_i) = U_i$. But no word in L has such a prefix, a contradiction. Thus, $U_1 V_1^{\omega} \cup \cdots \cup U_n V_n^{\omega} \neq L$. ◀

3.1.2 Characterization of Büchi Parikh Automata

As mentioned in the last section, the class of ω-languages recognized by Büchi PA is a strict subset of $\mathcal{L}^{\omega}_{\mathrm{PA,PA}}$, i. e., languages of the form $\bigcup_i U_i V_i^{\omega}$ for Parikh recognizable U_i and V_i. In this subsection we show that a restriction of the PA recognizing the V_i is sufficient to exactly capture the expressiveness of Büchi PA. To be precise, we show the following.

Lemma 3.1.6. *The following are equivalent for all ω-languages $L \subseteq \Sigma^{\omega}$.*

(1) L is Büchi PA recognizable.

(2) L is of the form $\bigcup_i U_i V_i^{\omega}$, where $U_i \in \Sigma^*$ is Parikh recognizable and $V_i \in \Sigma^*$ is recognized by a normalized PA where C is a homogeneous linear set.

We note that we can translate every PA (with a linear set C) into an equivalent normalized PA by Lemma 3.1.3. However, this construction adds a base vector, as we concatenate $\{1\}$ to C. In fact, this can generally not be avoided without losing expressiveness. It turns out that this loss of expressiveness is exactly what we need to characterize the class of ω-languages recognized by Büchi PA as stated in the previous lemma. The main reason for this is pointed out in the following lemma.

Lemma 3.1.7. *Let L be a language recognized by a (normalized) PA $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \{q_0\}, C)$ where C is a homogeneous linear set. Then we have $B_{\omega}(\mathcal{A}) = L(\mathcal{A})^{\omega}$.*

Proof. In this proof we assume that $C = C(\mathbf{0}, P)$ with $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_\ell\}$ for some $\ell \geq 0$.

Let $\alpha \in B_{\omega}(\mathcal{A})$ with accepting run $r = r_1 r_2 r_3 \ldots$ where $r_i = (p_{i-1}, \alpha_i, \mathbf{v}_i, p_i)$. As r satisfies the Büchi condition and $\mathcal{A}$ is normalized there are infinitely many accepting hits, that is, infinitely many i such that $p_i = q_0$ and $\rho(r_1 \ldots r_i) \in C$. By Dickson's Lemma [Dic13], there is an infinite monotone (sub)sequence of accepting hits $s_1 < s_2 < \ldots$, i. e., for all $j > i$ we have $\rho(r_1 \ldots r_{s_i}) = \mathbf{p}_1 z_1 + \cdots + \mathbf{p}_\ell z_\ell$ for some $z_i \in \mathbb{N}$ and $\rho(r_1 \ldots r_{s_j}) = \mathbf{p}_1 z'_1 + \cdots + \mathbf{p}_\ell z'_\ell$ for some $z'_i \in \mathbb{N}$, and $z'_k \geq z_k$ for all $k \leq \ell$. Hence, every infix $\alpha[s_i + 1, s_{i+1}]$ for $i \geq 0$ (assuming $s_0 = 0$) is accepted by $\mathcal{A}$.

Now let $w_1 w_2 \cdots \in L(\mathcal{A})^{\omega}$ such that $w_i \in L(\mathcal{A})$ for all $i \geq 1$. Let $r^{(i)}$ be an accepting run of $\mathcal{A}$ on w_i. Observe that for every $i \geq 1$ we have that $r^{(1)} \ldots r^{(i)}$ is an accepting run of $\mathcal{A}$ on $w_1 \ldots w_i$, as C is a homogeneous linear set, and hence we have $\rho(r^{(1)} \ldots r^{(i)}) = \rho(r^{(1)}) + \cdots + \rho(r^{(i)}) \in C$. Hence, the infinite sequence $r^{(1)} r^{(2)} \ldots$ is a run of $\mathcal{A}$ on $w_1 w_2 \ldots$ with infinitely many accepting hits. Hence $w_1 w_2 \cdots \in B_{\omega}(\mathcal{A})$. ◀

This is the main ingredient to prove Lemma 3.1.6.

Proof of Lemma 3.1.6. We note that the proof in [GJLZ22] showing that every ω-language L recognized by a Büchi-PA is of the form $\bigcup_i U_i V_i$ for PA recognizable U_i and V_i already constructs PA for the V_i of the desired form. This shows the implication (1) $\Rightarrow$ (2).

To show the implication (2) $\Rightarrow$ (1), we use that the ω-closure of languages recognized by PA of the stated form is Büchi PA recognizable by Lemma 3.1.7. As Büchi PA are closed under left-concatenation with PA recognizable languages (Lemma 3.1.2) and union [GJLZ22], the claim follows. ◀

3.1.3 Characterization of $\mathcal{L}^{\omega}_{\mathbf{PA,Reg}}$

In this subsection we characterize $\mathcal{L}^{\omega}_{\mathrm{PA,Reg}}$ by showing the following equivalences.

Theorem 3.1.8. *The following are equivalent for all ω-languages $L \subseteq \Sigma^{\omega}$.*

(1) L is of the form $\bigcup_i U_i V_i^{\omega}$, where $U_i \in \Sigma^*$ is Parikh recognizable, and $V_i \subseteq \Sigma^*$ is regular.

(2) L is limit PA recognizable.

(3) L is reachability-regular PA recognizable.

Observe that in the first item we may assume that L is of the form $\bigcup_i U_i V_i$, where $U_i \in \Sigma^*$ is Parikh recognizable, and $V_i \subseteq \Sigma^{\omega}$ is ω-regular. Then, by simple combinatorics and Büchi's theorem we have $\bigcup_i U_i V_i = \bigcup_i U_i(\bigcup_{j_i} X_{j_i} Y_{j_i}^{\omega}) = \bigcup_{i,j_i} U_i(X_{j_i} Y_{j_i}^{\omega}) = \bigcup_{i,j_i}(U_i X_{j_i}) Y_{j_i}^{\omega}$, for regular languages X_{j_i}, Y_{j_i}, where $U_i X_{j_i}$ is Parikh recognizable, as Parikh recognizable languages are closed under concatenation [Cad13, Proposition 3].

To simplify the proof, it is convenient to consider the following generalizations of Büchi automata. A *transition-based generalized Büchi automaton* (TGBA) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \mathcal{T})$ where $\mathcal{T} \subseteq 2^{\Delta}$ is a collection of sets of transitions. Then a run $r_1 r_2 r_3 \dots$ of $\mathcal{A}$ is accepting if for all $T \in \mathcal{T}$ there are infinitely many i such that $r_i \in T$. It is well-known that TGBA have the same expressiveness as Büchi automata [GL02].

Theorem 3.1.8 will be a direct consequence from the following lemmas. The first lemma shows the implication (1) $\Rightarrow$ (2).

Lemma 3.1.9. *If $L \in \mathcal{L}^{\omega}_{\mathrm{PA,Reg}}$, then L is limit PA recognizable.*

Proof. As the class of limit Parikh recognizable ω-languages is closed under union by Lemma 3.1.1, it is sufficient to show how to construct a limit PA for an ω-language of the form $L = UV^{\omega}$, where U is Parikh recognizable and V is regular.

Let $\mathcal{A}_1 = (Q_1, \Sigma, q_1, \Delta_1, F_1, C)$ be a PA with $L(\mathcal{A}_1) = U$ and $\mathcal{A}_2 = (Q_2, \Sigma, q_2, \Delta_2, F_2)$ be a Büchi automaton with $L_{\omega}(\mathcal{A}_2) = V^{\omega}$. We use the following standard construction for concatenation. Let $\mathcal{A} = (Q_1 \cup Q_2, \Sigma, q_1, \Delta, F_2, C)$ be a limit PA where

$$\Delta = \Delta_1 \cup \{(p, a, \mathbf{0}, q) \mid (p, a, q) \in \Delta_2\} \cup \{(f, a, \mathbf{0}, q) \mid (q_2, a, q) \in \Delta_2, f \in F_1\}.$$

We claim that $L_{\omega}(\mathcal{A}) = L$.

To show $L_\omega(\mathcal{A}) \subseteq L$, let $\alpha \in L_\omega(\mathcal{A})$ with accepting run $r_1 r_2 r_3 \dots$ where $r_i = (p_{i-1}, \alpha_i, \mathbf{v}_i, p_i)$. As only the states in F_2 are accepting, there is a position j such that $p_{j-1} \in F_1$ and $p_j \in Q_2$. In particular, all transitions of the copy of $\mathcal{A}_2$ are labeled with $\mathbf{0}$, i. e., $\mathbf{v}_i = \mathbf{0}$ for all $i \geq j$. Hence $\rho(r) = \rho(r_1 \dots r_{j-1}) \in C$ (in particular, there is no ∞ value in $\rho(r)$). We observe that $r_1 \dots r_{j-1}$ is an accepting run of $\mathcal{A}_1$ on $\alpha[1, j-1]$, as $p_{j-1} \in F_1$ and $\rho(r_1 \dots r_{j-1}) \in C$. For all $i \geq j$ let $r'_i = (p_{i-1}, \alpha_i, p_i)$. Observe that $(q_2, \alpha_j, p_j) r'_{j+1} r'_{j+2} \dots$ is an accepting run of $\mathcal{A}_2$ on $\alpha_j \alpha_{j+1} \alpha_{j+2} \dots$, hence $\alpha \in L(\mathcal{A}_1) \cdot L_\omega(\mathcal{A}_2) = L$.

To show $L = UV^\omega \subseteq L_\omega(\mathcal{A})$, let $w \in L(\mathcal{A}_1) = U$ with accepting run s, and $\alpha \in L_\omega(\mathcal{A}_2) = V^\omega$ with accepting run $r = r_1 r_2 r_3 \dots$, where $r_i = (p_{i-1}, \alpha_1, p_i)$. Observe that s is also a partial run of $\mathcal{A}$ on w, ending in an accepting state f. By definition of Δ, we can continue the run s in $\mathcal{A}$ basically as in r. To be precise, let $r'_1 = (f, \alpha_1, \mathbf{0}, p_1)$, and, for all $i > 1$ let $r'_i = (p_{i-1}, \alpha_i, \mathbf{0}, p_i)$. Then $s r'_1 r'_2 r'_3 \dots$ is an accepting run of $\mathcal{A}$ on $w\alpha$, hence $w\alpha \in L_\omega(\mathcal{A})$. ◀

Observe that the construction in the proof of the lemma works the same way when we interpret $\mathcal{A}$ as a reachability-regular PA (every visit of an accepting state has the same good counter value; this argument is even true if we interpret $\mathcal{A}$ as a Büchi PA), showing the implication (1) $\Rightarrow$ (3).

Corollary 3.1.10. *If $L \in \mathcal{L}^\omega_{\mathsf{PA,Reg}}$, then L is reachability-regular.*

For the backwards direction we need an auxiliary lemma, essentially stating that semi-linear sets over $C \subseteq (\mathbb{N} \cup \{\infty\})^d$ can be modified such that ∞-entries in vectors in C are replaced by arbitrary integers, and remain semi-linear.

Lemma 3.1.11. *Let $C \subseteq (\mathbb{N} \cup \{\infty\})^d$ be semi-linear and $D \subseteq \{1, \dots, d\}$. Let $C_D \subseteq \mathbb{N}^d$ be the set obtained from C as follows.*

1. *Remove every vector $\mathbf{v} = (v_1, \dots, v_d)$ where $v_i = \infty$ for an $i \notin D$.*
2. *As long as C_D contains a vector $\mathbf{v} = (v_1, \dots, v_d)$ with $v_i = \infty$ for an $i \leq d$: replace $\mathbf{v}$ by all vectors of the form $(v_1, \dots v_{i-1}, z, v_{i+1}, \dots, v_d)$ for $z \in \mathbb{N}$.*

Then C_D is semi-linear. Furthermore, C_D can be computed in polynomial time.

Proof. For a vector $\mathbf{v} = (v_1, \dots, v_d) \in (\mathbb{N} \cup \{\infty\})^d$, let $\mathsf{Inf}(\mathbf{v}) = \{i \mid v_i = \infty\}$ denote the positions of ∞-entries in $\mathbf{v}$. Furthermore, let $\bar{\mathbf{v}} = (\bar{v}_1, \dots, \bar{v}_d)$ denote the vector obtained from $\mathbf{v}$ by replacing every ∞-entry by 0, i. e., $\bar{v}_i = 0$ if $v_i = \infty$, and $\bar{v}_i = v_i$ otherwise.

We carry out the following procedure for every linear set of the semi-linear set independently, hence we assume that $C = (\mathbf{b}, P)$ with is linear. We also assume that there is no $\mathbf{p} \in P$ with $\mathsf{Inf}(\mathbf{p}) \not\subseteq D$, otherwise, we simply remove it.

Now, if $\mathsf{Inf}(\mathbf{b}) \not\subseteq D$, then $C_D = \emptyset$, as this implies that every vector in C has an ∞-entry at an unwanted position (the first item of the lemma).

Otherwise, $C_D = C(\mathbf{b}, \bigcup_{\mathbf{p}\in P}(\{\bar{\mathbf{p}}\} \cup \{\mathbf{e}_i \mid i \in \mathsf{Inf}(\mathbf{p})\}))$, which is linear by definition and computable in polynomial time. ◀

We are now ready to prove the following lemma, showing the implication (2) ⇒ (1).

Lemma 3.1.12. *If L is limit PA recognizable, then $L \in \mathcal{L}^{\omega}_{\mathsf{PA,Reg}}$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be an limit PA of dimension d. The idea is as follows. We guess a subset $D \subseteq \{1, \ldots, d\}$ of counters whose values we expect to be ∞. Observe that every counter not in D has a finite value, hence for every such counter there is a point where all transitions do not increment the counter further. For every subset $D \subseteq \{1, \ldots, d\}$ we decompose $\mathcal{A}$ into a PA and a TGBA. In the first step we construct a PA where every counter not in D reaches its final value and is verified. In the second step we construct a TGBA ensuring that for every counter in D at least one transition adding a non-zero value to that counter is used infinitely often. This can be encoded directly into the TGBA. Furthermore we delete all transitions that modify counters not in D.

Fix $D \subseteq \{1, \ldots, d\}$ and $f \in F$, and define the PA $\mathcal{A}^D_f = (Q, \Sigma, q_0, \Delta, \{f\}, C_D)$ where C_D is defined as in Lemma 3.1.11. Furthermore, we define the TGBA $\mathcal{B}^D_f = (Q, \Sigma, f, \Delta^D, \mathcal{T}^D)$ where Δ^D contains the subset of transitions of Δ where the counters not in D have zero-values (just the transitions without vectors for the counters, as we construct a TGBA). On the other hand, for every counter i in D there is one acceptance component in $\mathcal{T}^D$ that contains exactly those transitions (again without vectors) where the ith counter has a non-zero value. Finally, we encode the condition that at least one accepting state in F needs to by seen infinitely often in $\mathcal{T}^D$ by further adding the component $\{(p, a, q) \in \Delta \mid q \in F\}$ (i. e. now we need to see an incoming transition of a state in F infinitely often).

We claim that $L_\omega(\mathcal{A}) = \bigcup_{D\subseteq\{1,\ldots,d\}, f\in F} L(\mathcal{A}^D_f) \cdot L_\omega(\mathcal{B}^D_f)$, which by the comment below Theorem 3.1.8 and the equivalence of TGBA and Büchi automata implies the statement of the lemma.

To show $L_\omega(\mathcal{A}) \subseteq \bigcup_{D\subseteq\{1,\ldots,d\}, f\in F} L(\mathcal{A}^D_f) \cdot L_\omega(\mathcal{B}^D_f)$, let $\alpha \in L_\omega(\mathcal{A})$ with accepting run $r_1 r_2 r_3 \ldots$ where $r_i = (p_{i-1}, \alpha_i, \mathbf{v}_i, p_i)$. Let D be the positions of ∞-entries in $\rho(r) = (v_1, \ldots, v_d)$. As the v_i with $i \notin D$ have integer values, there is a position j such that in all $\mathbf{v}_k$ for $k \geq j$ the i-th entry of $\mathbf{v}_k$ is 0. Let $\ell \geq j$ be minimal such that p_ℓ in F. We split $\alpha = w\beta$, where $w = \alpha[1, \ell]$, and $\beta = \alpha_{\ell+1}\alpha_{\ell+2} \ldots$.

First we argue that $w \in L_\omega(\mathcal{A}^D_{p_\ell})$. Observe that $\mathcal{A}^D_{p_\ell}$ inherits all transitions from $\mathcal{A}$, hence $r_1 \ldots r_\ell$ is a run of $\mathcal{A}^D_{p_\ell}$ on w. As p_ℓ is accepting by definition, it remains to show that $\rho(r_1 \ldots r_\ell) \in C_D$. By the choice of ℓ, all counters not in D have reached their final values. As C_D contains all vectors of C where all ∞-entries are replaced by arbitrary values, the claim follows, hence $w \in L(\mathcal{A}^D_{p_\ell})$.

Now we argue that $\beta \in L_\omega(\mathcal{B}^D_{p_\ell})$. For every $k > \ell$ define $r'_k = (p_{k-1}, \alpha_k, p_k)$. Observe that $r' = r'_{k+1}r'_{k+2}\dots$ is a run of $\mathcal{B}^D_{p_\ell}$ on β (all r'_{k+1} exist in $\mathcal{B}^D_{p_\ell}$, as the counters not in D of all transitions r_k have zero-values by the definition of ℓ). It remains to show that r' is accepting, i. e., that for every counter in D at least one transition with a non-zero value is used infinitely often, and an accepting state is visited infinitely often. This is the case, as these counter values are ∞ in $\rho(r)$ and by the acceptance condition of limit PA, hence $\beta \in L_\omega(\mathcal{B}^D_{p_\ell})$. We conclude $\alpha \in \bigcup_{D\subseteq\{1,\dots,d\}, f\in F} L(\mathcal{A}^D_f) \cdot L_\omega(\mathcal{B}^D_f)$.

To show $\bigcup_{D\subseteq\{1,\dots,d\}, f\in F} L(\mathcal{A}^D_f) \cdot L_\omega(\mathcal{B}^D_f) \subseteq L_\omega(\mathcal{A})$, let $w \in L(\mathcal{A}^D_f)$ and $\beta \in L_\omega(\mathcal{B}^D_f)$ for some $D \subseteq \{1,\dots,d\}$ and $f \in F$. We show that $w\beta \in L_\omega(\mathcal{A})$.

Let s be an accepting run of $\mathcal{A}^D_f$ on w, which ends in the accepting state f by definition. Let $\rho(s) = (v_1, \dots, v_d)$. By definition of C_D, there is a vector $\mathbf{u} = (u_1, \dots, u_d)$ in C where $u_i = \infty$ if $i \in D$, and $u_i = v_i$ if $i \notin D$. Furthermore, let $r = r_1r_2r_3\dots$, where $r_i = (p_{i-1}, \alpha_i, p_i)$, be an accepting run of $\mathcal{B}^D_f$ on β, which starts in the accepting state f by definition. By definition of $\mathcal{T}^d$, for every counter $i \in D$ at least one transition where the i-th counter of the corresponding transition in Δ is non-zero is used infinitely often. Hence, let $r' = r'_1r'_2r'_3\dots$ where $r'_i = (p_{i-1}, \alpha_i, \mathbf{v}_i, p_i)$ for a suitable vector $\mathbf{v}_i$. Furthermore, the labels of transitions of counters not in D have a value of zero, hence $\rho(r') = (x_1, \dots, x_d)$, where $x_i = \infty$ if $i \in D$, and $x_i = 0$ if $i \notin D$. A technical remark: it might be the case that there are more than one transitions in Δ that collapse to the same transition in Δ^D, say $\delta_1 = (p, a, \mathbf{u}, q)$ and $\delta_2 = (p, a, \mathbf{v}, q)$ appear in Δ and collapse to (p, a, q) in Δ^D. If both transitions, δ_1 and δ_2, are seen infinitely often, we need to take care that we also see both infinitely often when translating the run r back. This is possible using a round-robin procedure.

Now observe that sr' is a run of $\mathcal{A}$ on $w\beta$ (recall that s ends in f, and r' starts in f). Furthermore, we have $\rho(sr') = \rho(s)+\rho(r') = (v_1+x_1, \dots, v_d+x_d)$, where $v_i+x_i = \infty$ if $i \in D$, and $v_i + x_i = v_i$ if $i \notin D$ by the observations above. Hence $\rho(sr') \in C$. Finally, $\mathcal{T}^D$ enforces that at least one accepting state in $\mathcal{B}^D_f$ is seen infinitely often, hence $w\beta \in L_\omega(\mathcal{A})$. ◀

Observe that the construction in Lemma 3.1.9 yields a limit PA whose semi-linear set C contains no vector with an ∞-entry. Hence, by this observation and the construction in the previous lemma we obtain the following corollary.

Corollary 3.1.13. *For every limit PA there is an equivalent limit PA whose semi-linear set does not contain any ∞-entries.*

Finally we show the implication (3) $\Rightarrow$ (1).

Lemma 3.1.14. *If L is reachability-regular, then $L \in \mathcal{L}^\omega_{\mathsf{PA,Reg}}$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a reachability-regular PA. The intuition is as follows. a reachability-regular PA just needs to verify the counters a single time. Hence, we can

recognize the prefixes of infinite words $\alpha \in B_\omega(\mathcal{A})$ that generate the accepting hit with a PA. Further checking that an accepting state is seen infinitely often can be done with a Büchi automaton.

Fix $f \in F$ and let $\mathcal{A}_f = (Q, \Sigma, q_0, \Delta, \{f\}, C)$ be the PA that is, syntactically equal to $\mathcal{A}$ with the only difference that f is the only accepting state. Similarly, let $\mathcal{B}_f = (Q, \Sigma, f, \{(p, a, q) \mid (p, a, \mathbf{v}, q) \in \Delta\}, F)$ be the Büchi automaton obtained from $\mathcal{A}$ by setting f as the initial state and the forgetting the vector labels.

We claim that $RR_\omega(\mathcal{A}) = \bigcup_{f \in F} L(\mathcal{A}_f) \cdot L_\omega(\mathcal{B}_f)$.

To show $RR_\omega(\mathcal{A}) \subseteq \bigcup_{f \in F} L(\mathcal{A}_f) \cdot L_\omega(\mathcal{B}_f)$, let $\alpha \in B_\omega(\mathcal{A})$ with accepting run $r = r_1 r_2 r_3 \dots$ where $r_i = (p_{i-1}, \alpha_i, \mathbf{v}_i, p_i)$. Let k be arbitrary such that there is an accepting hit in r_k (such a k exists by definition) and consider the prefix $\alpha[1, k]$. Obviously $r_1 \dots r_k$ is an accepting run of $\mathcal{A}_{p_k}$ on $\alpha[1, k]$. Furthermore, there are infinitely many j such that $p_j \in F$ by definition. In particular, there are also infinitely many $j \geq k$ with this property. Let $r'_i = (p_{i-1}, \alpha_i, p_i)$ for all $i > k$. Then $r'_{k+1} r'_{k+2} \dots$ is an accepting run of $\mathcal{B}_{p_k}$ on $\alpha_{k+1}\alpha_{k+2} \dots$ (recall that p_k is the initial state of $\mathcal{B}_{p_k}$). Hence we have $\alpha[1, k] \in L(\mathcal{A}_{p_k})$ and $\alpha_{k+1}\alpha_{k+2} \dots \in L_\omega(\mathcal{B}_{p_k})$.

To show $\bigcup_{f \in F} L(\mathcal{A}_f) \cdot L_\omega(\mathcal{B}_f) \subseteq RR_\omega(\mathcal{A})$, let $w \in L(\mathcal{A}_f)$ and $\beta \in L_\omega(\mathcal{B}_f)$ for some $f \in F$. We show $w\beta \in B_\omega(\mathcal{A})$. Let $s = s_1 \dots s_n$ be an accepting run of $\mathcal{A}_f$ on w, which ends in the accepting state f with $\rho(s) \in C$ by definition. Furthermore, let $r = r_1 r_2 r_3 \dots$ be an accepting run of $\mathcal{B}_f^D$ on β which starts in the accepting state f by definition. It is now easily verified that sr' with $r' = r'_1 r'_2 r'_3 \dots$ where $r'_i = (p_{i-1}, \alpha_i, \mathbf{v}_i, p_i)$ (for an arbitrary $\mathbf{v}_i$ such that $r'_i \in \Delta$) is an accepting run of $\mathcal{A}$ on $w\beta$, as there is an accepting hit in s_n, and the (infinitely many) visits of an accepting state in r translate one-to-one, hence $w\beta \in B_\omega(\mathcal{A})$. ◀

As shown in Lemma 3.1.6, the class of Büchi PA recognizable ω-languages is equivalent to the class of ω-languages of the form $\bigcup_i U_i V_i^\omega$ where U_i and V_i are Parikh recognizable, but the PA for V_i is restricted in such a way that the initial state is the only accepting state and the set is a homogeneous linear set. Observe that for every regular language L there is a Büchi automaton $\mathcal{A}$ where the initial state is the only accepting state with $L_\omega(\mathcal{A}) = L^\omega$ (see e. g., [Tho91, Lemma 1.2]). Hence, $\mathcal{L}^\omega_{\mathrm{PA,Reg}}$ is a subset of the class of Büchi PA recognizable ω-languages. This inclusion is also strict, as witnessed by the Büchi PA in Example 3.0.1 which has the mentioned property.

Corollary 3.1.15. *The class $\mathcal{L}^\omega_{\mathrm{PA,Reg}}$ is a strict subclass of the class of Büchi PA recognizable ω-languages.*

We finish this subsection by observing that reachability PA capture a subclass of $\mathcal{L}^\omega_{\mathrm{PA,Reg}}$ where, due to completeness, all $V_i = \Sigma$.

Observation 3.1.16. *The following are equivalent for all ω-languages $L \subseteq \Sigma^\omega$.*

(1) L is of the form $\bigcup_i U_i \Sigma^\omega$ where $U_i \subseteq \Sigma^*$ is Parikh recognizable.

(2) L is reachability PA recognizable.

3.1.4 Characterization of $\mathcal{L}^{\omega}_{\mathsf{PA,PA}}$ and $\mathcal{L}^{\omega}_{\mathsf{Reg,PA}}$

In this section we give a characterization of $\mathcal{L}^{\omega}_{\mathsf{PA,PA}}$ and a characterization of $\mathcal{L}^{\omega}_{\mathsf{Reg,PA}}$. As mentioned in the beginning of this section, reset PA are too strong to capture this class. However, restrictions of strong reset PA are good candidates to capture $\mathcal{L}^{\omega}_{\mathsf{PA,PA}}$ as well as $\mathcal{L}^{\omega}_{\mathsf{Reg,PA}}$. In fact we show that it is sufficient to restrict the appearances of accepting states to capture $\mathcal{L}^{\omega}_{\mathsf{PA,PA}}$, as specified by the first theorem of this subsection. Further restricting the vectors yields a model capturing $\mathcal{L}^{\omega}_{\mathsf{Reg,PA}}$, as specified in the second theorem of this subsection. Recall that the condensation of $\mathcal{A}$ is the DAG of strong components of the underlying graph of $\mathcal{A}$.

Theorem 3.1.17. *The following are equivalent for all ω-languages $L \subseteq \Sigma^{\omega}$.*

(1) $L \in \mathcal{L}^{\omega}_{\mathsf{PA,PA}}$.

(2) *L is recognized by a strong reset PA $\mathcal{A}$ with the property that accepting states appear only in the leaves of the condensation of $\mathcal{A}$, and there is at most one accepting state per leaf.*

Proof. To show the implication (1) $\Rightarrow$ (2), let $\mathcal{A}_i = (Q_i, \Sigma, q_i, \Delta_i, F_i)$ for $i \in \{1, 2\}$ be PA and let $L = L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)^{\omega}$. By Lemma 3.1.3 we may assume that $\mathcal{A}_2$ is normalized (recall that by Corollary 3.1.4 this implies $SR_{\omega}(\mathcal{A}_2) = L(\mathcal{A}_2)^{\omega}$) and hence write $L = L(\mathcal{A}_1) \cdot SR_{\omega}(\mathcal{A}_2)$. As pointed out in the proof of Lemma 3.1.2, we can construct a reset PA $\mathcal{A}$ that recognizes L such that only the accepting states of $\mathcal{A}_2$ remain accepting in $\mathcal{A}$. As $\mathcal{A}_2$ is normalized, this means that only q_2 is accepting in $\mathcal{A}$. Hence $\mathcal{A}$ satisfies the property of the theorem. Finally observe that the construction in Lemma 3.1.1 maintains this property, implying that the construction presented in Lemma 3.1.5 always yields a reset PA of the desired form.

To show the implication (2) $\Rightarrow$ (1), let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a strong reset PA of dimension d with the mentioned property. Let $f \in F$ and let $\mathcal{A}_f = (Q, \Sigma, q_0, \Delta_f, \{f\}, C \cdot \{1\})$ with $\Delta_f = \{p, a, \mathbf{v} \cdot 0, q) \mid (p, a, \mathbf{v}, q) \in \Delta, q \neq f\} \cup \{(p, a, \mathbf{v} \cdot 1, f) \mid (p, a, \mathbf{v}, f) \in \Delta\}$ be the PA of dimension $d + 1$ obtained from $\mathcal{A}$ by setting f as the only accepting state with an additional counter that is 0 at every transition except the incoming transitions of f, where the counter is set to 1. Additionally all vectors in C are concatenated with 1. Similarly, let $\mathcal{A}_{f,f} = (Q, \Sigma, f, \Delta_f, \{f\}, C \cdot \{1\})$ be the PA of dimension $d+1$ obtained from $\mathcal{A}_f$ by setting f as the initial state.

To show $SR_{\omega}(\mathcal{A}) \subseteq \bigcup_{f \in F} L(\mathcal{A}_f) \cdot L(\mathcal{A}_{f,f})^{\omega}$, let $\alpha \in S_{\omega}(\mathcal{A})$ with accepting run $r = r_1 r_2 r_3 \ldots$ where $r_i = (p_{i-1}, \alpha_i, \mathbf{v}_i, p_i)$. Let $k_1 < k_2 < \ldots$ be the positions of accepting states in r, i. e., $p_{k_i} \in F$ for all $i \geq 1$. First observe that the property in the theorem implies $p_{k_i} = p_{k_j}$ for all $i, j \geq 1$, i. e., no two distinct accepting states appear in r, since accepting states appear only in different leaves of the condensation of $\mathcal{A}$.

For $j \geq 1$ define $r'_j = (p_{j-1}, \alpha_j, \mathbf{v}_j \cdot 0, p_j)$ if $j \neq k_i$ for all $i \geq 1$, and $r'_j = (p_{j-1}, \alpha_j, \mathbf{v}_j \cdot 1, p_j)$ if $j = k_i$ for some $i \geq 1$, i. e., we replace every transition r_j by the matching transition in Δ_f.

Now consider the partial run $r_1 \dots r_{k_1}$ and observe that $p_i \neq p_{k_1}$ for all $i < k_1$, and $\rho(r_1 \dots r_{k_1}) \in C$ by the definition of strong reset PA. Hence $r' = r'_1 \dots r'_{k_1}$ is an accepting run of $\mathcal{A}_{p_{k_1}}$ on $\alpha[1, k_1]$, as only a single accepting state appears in r', the newly introduced counter has a value of 1 when entering p_{k_1}, i. e., $\rho(r') \in C \cdot \{1\}$, hence $\alpha[1, k_1] \in L(\mathcal{A}_{p_{k_1}})$.

Finally, we show that $\alpha[k_i + 1, k_{i+1}] \in L(\mathcal{A}_{p_{k_1}, p_{k_1}})$. Observe that $r'_{k_i+1} \dots r'_{k_{i+1}}$ is an accepting run of $\mathcal{A}_{p_{k_1}, p_{k_1}}$ on $\alpha[k_i + 1, k_{i+1}]$: we have $\rho(r_{k_i+1} \dots r_{k_{i+1}}) = \mathbf{v} \in C$ by definition. Again, as only a single accepting state appears in $r'_{k_i+1} \dots r'_{k_{i+1}}$, we have $\rho(r'_{k_i+1} \dots r'_{k_{i+1}}) = \mathbf{v} \cdot 1 \in C \cdot \{1\}$, and hence $\alpha[k_i + 1, k_{i+1}] \in L(\mathcal{A}_{p_{k_1}, p_{k_1}})$. We conclude $\alpha \in L(\mathcal{A}_{p_{k_1}}) \cdot L(\mathcal{A}_{p_{k_1}, p_{k_1}})^\omega$.

To show $\bigcup_{f \in F} L(\mathcal{A}_f) \cdot L(\mathcal{A}_{f,f})^\omega \subseteq SR_\omega(\mathcal{A})$, let $u \in L(\mathcal{A}_f)$, and $v_1, v_2, \dots \in L(\mathcal{A}_{f,f})$ for some $f \in F$. We show that $uv_1v_2 \dots \in SR_\omega(\mathcal{A})$.

First let $u = u_1 \dots u_n$ and $r' = r'_1 \dots r'_n$ with $r'_i = (p_{i-1}, u_i, \mathbf{v}_i \cdot c_i, p_i)$, where $c_i \in \{0, 1\}$, be an accepting run of $\mathcal{A}_f$ on u. Observe that $\rho(r') \in C \cdot \{1\}$, hence $\sum_{i \leq n} c_i = 1$, i. e., p_n is the only occurrence of an accepting state in r' (if there was another, say p_j, then $c_j = 1$ by the choice of Δ_f, hence $\sum_{i \leq n} c_i > 1$, a contradiction). For all $1 \leq i \leq n$ let $r_i = (p_{i-1}, u_i, \mathbf{v}_i, p_i)$. Then $r_1 \dots r_n$ is a partial run of $\mathcal{A}$ on w with $\rho(r_1 \dots r_n) \in C$ and $p_n = f$.

Similarly, no run of $\mathcal{A}_{f,f}$ on any v_i visits an accepting state before reading the last symbol, hence we continue the run from r_n on $v_1, v_2, \dots$ using the same argument. Hence $uv_1v_2 \dots \in SR_\omega(\mathcal{A})$, concluding the proof. ◀

As a side product of the proof of Theorem 3.1.17 we get the following corollary, which is in general not true for arbitrary reset PA.

Corollary 3.1.18. *Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a strong reset PA with the property that accepting states appear only in the leaves of the condensation of $\mathcal{A}$, and there is at most one accepting state per leaf. Then we have $SR_\omega(\mathcal{A}) = \bigcup_{f \in F} S_\omega(Q, \Sigma, q_0, \Delta, \{f\}, C)$.*

By even further restricting the power of strong reset PA, we get the following characterization of $\mathcal{L}^\omega_{\text{Reg,PA}}$.

Theorem 3.1.19. *The following are equivalent for all ω-languages $L \subseteq \Sigma^\omega$.*

(1) L is of the form $\bigcup_i U_i V_i^\omega$, where $U_i \subseteq \Sigma^*$ is regular and $V_i \subseteq \Sigma^*$ is Parikh recognizable.

(2) L is recognized by a strong reset PA $\mathcal{A}$ with the following properties.

(a) At most one state q per leaf of the condensation of $\mathcal{A}$ may have incoming transitions from outside the leaf, this state q is the only accepting state in the leaf, and there are no accepting states in non-leaves.

(b) only transitions connecting states in a leaf may be labeled with a non-zero vector.

Observe that property (a) is a stronger property than the one of Theorem 3.1.17, hence, strong reset PA with this restriction are at most as powerful as those that characterize $\mathcal{L}^{\omega}_{\mathsf{PA},\mathsf{PA}}$. However, as a side product of the proof we get that property (a) is equivalent to the property of Theorem 3.1.17. Hence, property (b) is necessary to sufficiently weaken strong reset PA such that they capture $\mathcal{L}^{\omega}_{\mathsf{Reg},\mathsf{PA}}$. In fact, using the notion of normalization, we can re-use most of the ideas in the proof of Theorem 3.1.17.

Proof of Theorem 3.1.19. We can trivially convert an NFA into an equivalent PA by labeling every transition with 0 and choosing $C = \{0\}$, showing the implication (1) $\Rightarrow$ (2). Let $\mathcal{A}$ be an arbitrary PA and assume that it is normalized; in particular implying that it is only a single SCC. Again, we have $L(\mathcal{A})^{\omega} = S_{\omega}(\mathcal{A})$ and the constructions for concatenation and union preserve the properties, hence, we obtain a strong reset PA of the desired form.

To show the implication (2) $\Rightarrow$ (1), let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a strong reset PA of dimension d with properties (a) and (b). Fix $f \in F$ and let

$$\mathcal{B}_f = (Q_f, \Sigma, q_0, \{(p, a, q) \mid (p, a, \mathbf{v}, q) \in \Delta, p, q \in Q_f\}, \{f\})$$

with $Q_f = \{q \in Q \mid q \text{ appears in a non-leaf SCC of } C(\mathcal{A})\} \cup \{f\}$ be the NFA obtained from $\mathcal{A}$ by removing all leaf states except f, and removing all labels from the transitions. Recycling the automaton from Theorem 3.1.17, let $\mathcal{A}_{f,f} = (Q, \Sigma, f, \Delta_f, \{f\}, C \cdot \{1\})$ with $\Delta_f = \{(p, a, \mathbf{v} \cdot 0, q) \mid (p, a, \mathbf{v}, q) \in \Delta, q \neq f\} \cup \{(p, a, \mathbf{v} \cdot 1, f) \mid (p, a, \mathbf{v}, f) \in \Delta\}$. We claim $SR_{\omega}(\mathcal{A}) = \bigcup_{f \in F} L(\mathcal{B}_f) \cdot L(\mathcal{A}_{f,f})^{\omega}$.

To show $SR_{\omega}(\mathcal{A}) \subseteq \bigcup_{f \in F} L(\mathcal{B}_f) \cdot L(\mathcal{A}_{f,f})^{\omega}$, let $\alpha \in SR_{\omega}(\mathcal{A})$ with accepting run $r = r_1 r_2 r_3 \ldots$ where $r_i = (p_{i-1}, \alpha_i, \mathbf{v}_i, p_i)$, and let $k_1 < k_2 < \ldots$ be the positions of the accepting states in r, and consider the partial run $r_1 \ldots r_{k_1}$ (if $k_1 = 0$, i. e., the initial state is already accepting, then $r_1 \ldots r_{k_1}$ is empty).

By property (a) we have that p_{k_1} is the first state visited in r that is, located in a leaf of $C(\mathcal{A})$. Hence $r'_1 \ldots r'_{k_1}$, where $r'_i = (p_{i-1}, \alpha_i, p_i)$, is an accepting run of $\mathcal{B}_{p_{k_1}}$ on $\alpha[1, k_1]$ (in the case $k_1 = 0$ we define $\alpha[1, k_1] = \varepsilon$).

By the same argument as in the proof of Theorem 3.1.17 we have $p_{k_i} = p_{k_j}$ for all $i, j \geq 1$, hence $\alpha[k_i + 1, k_{i+1}] \in L(\mathcal{A}_{p_{k_1}, p_{k_1}})$, and hence $\alpha \in L(\mathcal{B}_{p_k}) \cdot L(\mathcal{A}_{p_{k_1}, p_{k_1}})^{\omega}$.

To show $\bigcup_{f \in F} L(\mathcal{A}_f) \cdot L(\mathcal{A}_{f,f})^{\omega} \subseteq SR_{\omega}(\mathcal{A})$, let $u \in L(\mathcal{B}_f)$, and $v_1, v_2, \cdots \in L(\mathcal{A}_{f,f})$ for some $f \in F$. We show that $uv_1v_2 \cdots \in S_{\omega}(\mathcal{A})$.

First observe that properties (a) and (b) enforce that $\mathbf{0} \in C$, as the accepting state of a leaf of $C(\mathcal{A})$ is visited before a transition labeled with a non-zero can be used. Let $u = u_1 \ldots u_n$ and $s_1 \ldots s_n$ with $s_i = (p_{i_1}, u_i, p_i)$ be an accepting run of $\mathcal{B}_f$ on u. Define $s'_i = (p_{i_1}, u_i, \mathbf{0}, p_i)$ and observe that $s'_1 \ldots s'_n$ is a partial run of $\mathcal{A}$ with $\rho(s'_1 \ldots s'_n) \in C$ and $p_n = f$ by the observation above. Again we can very similarly continue the run on $v_1, v_2, \ldots$ using the same argument. Hence $uv_1v_2 \cdots \in SR_{\omega}(\mathcal{A})$, concluding the proof. ◀

3.1.5 Blind Counter Automata and ε-elimination

As mentioned in Section 1.2, blind multicounter machines and Parikh automata are equivalent on finite words. Blind counter automata on infinite words were first studied by Fernau and Stiebe [FS08]. In this section we first recall the definition of blind counter machines as introduced by Fernau and Stiebe [FS08]. The definition of these automata admits ε-transitions. It is easily observed that Büchi PA with ε-transitions are equivalent to blind counter machines. Therefore, we extend all models studied in this paper with ε-transitions and consider the natural question whether they admit ε-elimination. We show that almost all models allow ε-elimination, the exception being safety and co-Büchi PA. As it turns out, co-Büchi PA with ε-transitions are powerful enough to simulate Büchi PA, and safety PA with ε-transitions are powerful enough to simulate reset PA. This is a stark contrast to their variants without ε-transitions, as they do not even recognize all ω-regular languages.

In order to avoid confusion (and sticking to the notation in the literature), we use the terms blind multicounter machines (BMCM), meaning the model operating on finite words introduced in Chapter 2, while blind counter automata (BCA) always denotes the model operating on infinite words which we introduce now. Similar to Parikh automata, the syntax of blind counter automata operating on infinite words is identical to blind multicounter machines on finite words, that is, a tuple $\mathcal{M} = (Q, \Sigma, q_0, \Delta, F)$ where $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{Z}^d \times Q$ for some $d \geq 1$ is a finite set. We adapt the definition of configuration to infinite words as follows. A *configuration* of $\mathcal{M}$ is a tuple $c = (q, \alpha, \mathbf{v}) \in Q \times \Sigma^\omega \times \mathbb{Z}^d$, where q is the current state, α is the suffix of the input word that has not been read yet, and $\mathbf{v}$ represents the current counter value. We say c *derives* into c', written $c \vdash_{\mathcal{M}} c'$, if $c = (q, a\beta, \mathbf{v})$, $c' = (q', \beta, \mathbf{v}')$ and $(q, a, \mathbf{u}, q') \in \Delta$ with $\mathbf{v}' = \mathbf{v} + \mathbf{u}$; or if $c = (q, \beta, \mathbf{v})$, $c' = (q', \beta, \mathbf{v}')$ and $(q, \varepsilon, \mathbf{u}, q') \in \Delta$ with $\mathbf{v}' = \mathbf{v} + \mathbf{u}$. Wa say $\mathcal{M}$ *accepts* an infinite word α if there is an infinite sequence of configuration derivations $c_1 \vdash c_2 \vdash c_3 \vdash \dots$ with $c_1 = (q_0, \varepsilon, \alpha, \mathbf{0})$ such that for infinitely many i we have $c_i = (p_i, \alpha_{j+1}\alpha_{j+2} \dots, \mathbf{0})$ with $p_i \in F$ and for all $j \geq 1$ there is a configuration of the form $(p, \alpha_{j+1}\alpha_{j+2} \dots, \mathbf{v})$ for some $p \in Q$ and $\mathbf{v} \in \mathbb{Z}^k$ in the sequence. That is, an infinite word is accepted if we infinitely often visit an accepting state when the counters are $\mathbf{0}$, and every symbol of α is read at some point. We define the ω-language recognized by $\mathcal{M}$ as $L_\omega(\mathcal{M}) = \{\alpha \in \Sigma^\omega \mid \mathcal{M} \text{ accepts } \alpha\}$.

Parikh automata naturally generalize to Parikh automata with ε-transitions. An ε-PA is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \mathcal{E}, F, C)$ where $\mathcal{E} \subseteq Q \times \{\varepsilon\} \times \mathbb{N}^d \times Q$ is a finite set of *labeled ε-transitions*, and all other entries are defined as for PA. A run of $\mathcal{A}$ on an infinite word $\alpha_1\alpha_2\alpha_3 \dots$ is an infinite sequence of transitions $r \in (\mathcal{E}^*\Delta)^\omega$, say $r = r_1 r_2 r_3 \dots$ with $r_i = (p_{i-1}, \gamma_i, \mathbf{v}_i, p_i)$ such that $p_0 = q_0$, and $\gamma_i = \varepsilon$ if $r_i \in \mathcal{E}$, and $\gamma_i = \alpha_j$ if $r_i \in \Delta$ is the j-th occurrence of a (non-ε) transition in r. The acceptance conditions of the models translate to runs of ε-PA in the obvious way. We use terms like ε-safety PA, ε-reachability PA, etc, to denote an ε-PA with the respective acceptance condition.

Note that we can treat every PA as an ε-PA, that is, a PA $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ is equivalent to the ε-PA $\mathcal{A}' = (Q, \Sigma, q_0, \Delta, \emptyset, F, C)$.

3.1.6 Blind Counter Automata vs. Büchi Parikh Automata

We start with the following simple observation.

Lemma 3.1.20. *BCA and ε-Büchi PA are equivalent.*

Proof. We first show that for every BCA $\mathcal{M}$ there is an equivalent ε-Büchi PA $\mathcal{A}$. Let $\mathcal{M} = (Q, \Sigma, q_0, \Delta, F)$ be a BCA of dimension d. For a vector $(x_1, \ldots, x_d) \in \mathbb{Z}^d$ we define the vector $\mathbf{v}^{\pm} = (x_1^+, \ldots x_d^+, x_1^-, \ldots x_d^-) \in \mathbb{N}^{2d}$ as follows: if x_i is positive, then $x_i^+ = x_i$ and $x_i^- = 0$. Otherwise, $x_i^+ = 0$ and $x_i^- = |x_i|$. We construct an equivalent ε-Büchi PA $\mathcal{A} = (Q, \Sigma, q_0, \Delta', \mathcal{E}', F, C)$ of dimension $2d$, where $\Delta' = \{(p, a, \mathbf{v}^{\pm}, q) \mid (p, a, \mathbf{v}, q) \in \Delta\}$ and $\mathcal{E}' = \{(p, \varepsilon, \mathbf{v}^{\pm}, q) \mid (p, \varepsilon, \mathbf{v}, q) \in \Delta\}$. Finally, let $C = \{(x_1, \ldots, x_d, x_1, \ldots, x_d) \mid x_i \in \mathbb{N}\}$. It is now easily verified that $L_\omega(\mathcal{M}) = P_\omega(\mathcal{A})$.

For the reverse direction we show that for every Büchi PA $\mathcal{A}$ there is an equivalent BCA $\mathcal{M}$. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a Büchi PA of dimension d where $C = \bigcup_{i \leq \ell} C(\mathbf{b}_i, P_i)$. Note that we have $B_\omega(\mathcal{A}) = \bigcup_{i \leq \ell} B_\omega(Q, \Sigma, q_0, \Delta, F, C(\mathbf{b}_i, P_i))$ by the infinite pigeonhole principle. Hence, we can assume that $C = C(\mathbf{b}, P)$ is linear as BCA are closed under union [FS08]. We construct a blind d-counter machine $\mathcal{M}$ that simulates $\mathcal{A}$ as follows: $\mathcal{M}$ consists of a copy of $\mathcal{A}$ where the accepting states have are equipped with additional ε-transitions subtracting the period vectors $\mathbf{p} \in P$. We only need to consider the base vector $\mathbf{b}$ a single time, hence we introduce a fresh initial state q_0' and a ε-transition from q_0' to q_0 subtracting $\mathbf{b}$. Formally, we construct $\mathcal{M} = (Q \cup \{q_0'\}, \Sigma, q_0', \Delta', F)$ where

$$\Delta' = \Delta \cup \{(q_0', \varepsilon, -\mathbf{b}, q_0\} \cup \{(q_f, \varepsilon, -\mathbf{p}, q_f) \mid q_f \in F, \mathbf{p} \in P\}.$$

It is now easily verified that $B_\omega(\mathcal{A}) = L_\omega(\mathcal{M})$. ◀

3.1.7 ε-elimination

We now show that almost all PA models admit ε-elimination. We first consider Büchi PA, where ε-elimination implies the equivalence of blind counter machines and Büchi PA by Lemma 3.1.20. We provided an elementary combinatorial proof on the automaton level in the manuscript [GSS23, Theorem 4.2]. We thank Georg Zetzsche for outlining a much simpler proof (using a heavier toolbox) which we present here.

Theorem 3.1.21. *ε-Büchi PA admit ε-elimination.*

Proof. Observe that the construction in Lemma 3.1.20 translates ε-free BCA into ε-free Büchi PA. We can hence translate a given Büchi PA into a BCA and eliminate ε-transitions and then translate back into a Büchi PA. Therefore, all we need to show is that BCA admit

ε-elimination. To show that BCA admit ε-elimination we observe that

$$L \text{ is recognized by a BCA} \quad \Leftrightarrow \quad L = \bigcup_i U_i V_i^\omega,$$

where U_i is a language of finite words that is recognized by a BMCM (on finite words) and V_i is a language of finite words that is recognized by a BMCM where $F = \{q_0\}$. The proof of this observation is very similar to the proof of Lemma 3.1.6 and we leave the details to the reader.

As shown in [Gre78, Lat79, Zet13], BMCM admit ε-elimination. Furthermore, from the proof technique established in [Zet13, Lemma 7.7] it is immediate that the condition $F = \{q_0\}$ can be preserved. We obtain ε-free BMCM $\mathcal{A}_i'$ and $\mathcal{B}_i'$ for the languages U_i and V_i. Using the construction of [KR03b, Theorem 32], we can translate $\mathcal{A}_i'$ and $\mathcal{B}_i'$ into PA $\mathcal{A}_i$ and $\mathcal{B}_i$, where the $\mathcal{B}_i$ satisfy $F_i = \{q_0\}$ and the sets C_i are homogeneous linear sets. Now the statement follows by Lemma 3.1.6. ◀

We continue with ε-reachability, ε-reachability-regular and ε-limit PA, as we show ε-elimination using the same technique for these models. As shown in Observation 3.1.16 and Theorem 3.1.8, the class of ω-languages recognized by reachability PA coincides with the class of ω-languages of the form $\bigcup_i U_i \Sigma^\omega$ for Parikh recognizable U_i, and the class of reachability-regular and limit PA recognizable ω-languages coincides with the class of ω-languages of the form $\bigcup_i U_i V_i^\omega$ for Parikh recognizable U_i and regular V_i, respectively. It is well-known that NFA and PA on finite words are closed under homomorphisms and hence admit ε-elimination [KR03b] (as a consequence of [Lat79, Proposition II.11], ε-transitions can even be eliminated without changing the semi-linear set). The characterizations allow us to reduce ε-elimination of these infinite word PA to the finite case.

Lemma 3.1.22. *ε-reachability, ε-reachability-regular, and ε-limit PA admit ε-elimination.*

Proof. We show the statement for ε-reachability PA. The technique can very easily be translated to the other two models. Let $\mathcal{A}$ be an ε-reachability PA with $R_\omega(\mathcal{A}) = L \subseteq \Sigma^\omega$. Let $\mathcal{A}_e$ be the reachability PA obtained from $\mathcal{A}$ by replacing every ε-transition with an e-transition, where e is a fresh symbol that does not appear in Σ. Let h be the homomorphism that erases the symbol e, i.e., $h(e) = \varepsilon$. Observe that $\mathcal{A}_e$ recognizes an ω-language $L_e \subseteq (\Sigma \cup \{e\})^\omega$ with the property that $h(L_e) = L$ (note that by definition $\{\varepsilon\}^\omega = \emptyset$). Now, by Observation 3.1.16 we can write L_e as $\bigcup_i U_i \cdot (\Sigma \cup \{e\})^\omega$ where $U_i \subseteq (\Sigma \cup \{e\})^*$ is Parikh recognizable. As the class of Parikh recognizable languages is closed under homomorphisms [KR03b], we have

$$L = h(L_e) = h\left(\bigcup_i U_i \cdot (\Sigma \cup \{e\})^\omega\right) = \bigcup_i h(U_i) \cdot \Sigma^\omega,$$

and can hence find a reachability PA for L. The proof for reachability-regular and limit PA works the same way, as the regular languages are also closed under homomorphisms. ◀

Now we show that strong ε-reset PA and weak ε-reset PA admit ε-elimination. We show that these two models are equivalent. Hence to show this statement we only need to argue that strong ε-reset PA admit ε-elimination.

Lemma 3.1.23. *Every strong ε-reset PA $\mathcal{A}$ is equivalent to a weak ε-reset PA $\mathcal{A}'$ that has the same set of states and uses one additional counter. If $\mathcal{A}$ is a strong reset PA, then $\mathcal{A}'$ is a weak reset PA.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \mathcal{E}, F, C)$ be a strong ε-reset PA. We construct an equivalent weak ε-reset PA $\mathcal{A}'$ that simulates $\mathcal{A}$, ensuring that no run visits an accepting state without resetting. To achieve that, we add an additional counter that tracks the number of visits of an accepting state (without resetting). Moreover, we define $C' = C \cdot \{1\}$, such that this new counter must be set to 1 when visiting an accepting state, thus disallowing to pass such a state without resetting. Now it is clear that $\mathcal{A}'$ is a weak ε-reset PA equivalent to $\mathcal{A}$. Observe that if $\mathcal{A}$ has no ε-transitions, then $\mathcal{A}'$ has no ε-transitions. ◀

Lemma 3.1.24. *Every weak ε-reset PA $\mathcal{A}$ is equivalent to a strong ε-reset PA $\mathcal{A}'$ with at most twice the number of states and the same number of counters. If $\mathcal{A}$ is a strong reset PA, then $\mathcal{A}'$ is a weak reset PA.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \mathcal{E}, F, C)$ be a weak ε-reset PA. We construct an equivalent strong ε-reset PA $\mathcal{A}'$ that simulates $\mathcal{A}$ by having the option to "avoid" accepting states arbitrarily long. For this purpose, we create a non-accepting copy of F. Consequently, $\mathcal{A}'$ can decide to continue or reset a partial run using non-determinism. Again, it is clear that $\mathcal{A}'$ is equivalent to $\mathcal{A}$. Observe that if $\mathcal{A}$ has no ε-transitions, then $\mathcal{A}'$ has no ε-transitions. ◀

Lemma 3.1.25. *Strong ε-reset PA admit ε-elimination.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \mathcal{E}, F, C)$ be a strong ε-reset PA of dimension d. We assume without loss of generality that q_0 has no incoming transitions (this can be achieved by introducing a fresh copy of q_0). Furthermore, we assume that $F \neq \emptyset$ (otherwise $SR_\omega(\mathcal{A}) = \emptyset$). Let the states of Q be ordered arbitrarily, say $Q = \{q_0, \ldots, q_{n-1}\}$. We construct an equivalent strong reset PA $\mathcal{A}' = (Q', \Sigma, q_0, \Delta', F', C')$ of dimension $d + n$. In the beginning, $\mathcal{A}'$ is a copy of $\mathcal{A}$ (keeping the ε-transitions for now), which is modified step-by-step. The purpose of the new counters is to keep track of the states that have been visited (since the last reset). Initially, we hence modify the transitions as follows: for every transition $(q_i, \gamma, \mathbf{v}, q_j) \in \Delta \cup \mathcal{E}$ we replace $\mathbf{v}$ by $\mathbf{v} \cdot \mathbf{e}_j^n$.

Let $p, q \in Q$. Assume there is a sequence of transitions $\tilde{\lambda} = r_1 \dots r_j \dots r_k \in \mathcal{E}^* \Delta \mathcal{E}^*$; $1 \leq j \leq k \leq 2n+1$, where

- $r_j = (p_{j-1}, a, \mathbf{v}_j, p_j) \in \Delta$, and
- $r_i = (p_{i-1}, \varepsilon, \mathbf{v}_i, p_i) \in \mathcal{E}$ for all $i \neq j, i \leq k$,
- such that $p_0 = p, p_k = q$, and $p_i \neq p_\ell$ for $i, \ell \leq j$ and $p_i \neq p_\ell$ for $i, \ell \geq j$, and
- all internal states are non-accepting, i. e., $p_i \notin F$ for all $0 < i < k$.

Then we introduce the *shortcut* $(p, a, \rho(\tilde{\lambda}), q)$, where $\rho(\tilde{\lambda})$ is computed already with respect to the new counters, tracking that the p_i in $\tilde{\lambda}$ have been visited, i. e., the counters corresponding to the p_i in this sequence have non-zero values.

Let $p, q \in Q$. We call a (possibly empty) sequence $\lambda = r_1 \dots r_k \in \mathcal{E}^*$ with $r_i = (p_{i-1}, \varepsilon, \mathbf{v}_i, p_i)$ and $p_0 = p, p_k = q$ a *no-reset ε-sequence* from p to q if all internal states are non-accepting, i. e., $p_i \notin F$ for all $0 < i < k$. A *no-reset ε-path* is a no-reset sequence such that $p_i \neq p_j$ for $i \neq j$. Observe that the set of no-reset ε-paths from p to q is finite, as the length of each path is bounded by $n-1$. We call the pair (p, q) a *C-pair* if there is a no-reset ε-sequence r from p to q with $\rho(r) \in C$, where $\rho(r)$ is computed in $\mathcal{A}$.

Let $S = (f_1, \dots, f_\ell)$ be a non-empty sequence of pairwise distinct accepting states (note that this implies $\ell \leq n$). We call S a *C-sequence* if each (f_i, f_{i+1}) is a C-pair.

For all $p, q \in F$ and C-sequences S such that $p = f_1$ if $p \in F$ and $q = f_\ell$ if $q \in F$, we introduce a new state (p, S, q). We add (p, S, q) to F', that is, we make the new states accepting. State (p, S, q) will represent a partial run of the automaton with only ε-transitions starting in p, visiting the accepting states of S in that order, and ending in q.

Observe that in the following we introduce only finitely many transitions by the observations made above; we will not repeat this statement in each step. Let $p, q \in Q$ and $S = (f_1, \dots, f_\ell)$ be a C-sequence. For every transition of the form $(s, a, \mathbf{v}, p) \in \Delta$ we insert new transitions

$$\{(s, a, \mathbf{v} + \rho(\lambda), (p, S, q)) \mid \lambda \text{ is a no-reset } \varepsilon\text{-path from } p \text{ to } f_1\}$$

to Δ'. Similarly, for every transition of the form $(q, a, \mathbf{v}, t) \in \Delta$ we insert new transitions

$$\{((p, S, q), a, \mathbf{v} + \rho(\lambda), t) \mid \lambda \text{ is a no-reset } \varepsilon\text{-path from } f_\ell \text{ to } q\}$$

to Δ'. Again this set is finite. Additionally, let $p', q' \in Q$ and $S' = (f'_1, \dots, f'_k)$ be a C-sequence. For every sequence $\tilde{\lambda} = \lambda \delta \lambda'$ where λ is a no-reset ε-path from f_ℓ to q, $\delta = (q, a, \mathbf{v}, p')$, and λ' is a no-reset ε-path from p' to f'_1 we add the shortcuts $\big((p, S, q), a, \rho(\tilde{\lambda}), (p', S', q')\big)$ to Δ'.

Lastly, we connect the initial state q_0 in a similar way (recall that we assume that q_0 has no incoming transitions, and in particular no loops). For every transition $(p, a, \mathbf{v}, q) \in \Delta$ and every C-sequence $S = (f_1, \dots, f_l)$ with the property that (q_0, f_1) is a C-pair and there is a no-reset ε-path λ from f_ℓ to p, we introduce the transition $(q_0, a, \rho(\lambda) + \mathbf{v}, q)$ for every

such path λ. Additionally, for every C-sequence $S' = (f'_1, \dots f'_k)$ such that there is a no-reset ε-path λ' from q to f'_1, we introduce the transition $\big(q_0, a, \rho(\lambda) + \mathbf{v} + \rho(\lambda'), (q, S', t)\big)$ for all such paths λ, λ' and $t \in Q$. Furthermore, for every no-reset ε-path $\hat{\lambda}$ from q_0 to p, we introduce the transition $(q_0, a, \rho(\hat{\lambda}) + \mathbf{v} + \rho(\lambda'), (q, S', t))$ for all $t \in Q$. A reader who is worried that we may introduce too many transitions at this point shall recall that (q, S', t) has no outgoing transition if there does not exist a no-reset ε-path from f'_k to t. Finally, we delete all ε-transitions.

We define C' similar to the construction by Klaedtke and Ruess [KR03b] used to eliminate ε-transitions in the finite setting. For every $q \in Q \setminus F$ we define $C_q = \{\rho(r) \mid r \in \mathcal{E}^*$ is partial run of $\mathcal{A}$ starting and ending in q that does not visit any accepting state$\}$. As a consequence of Parikh's theorem [Par66] and [KR03b, Lemma 5], the sets C_q are semi-linear. Then $C' = \{\mathbf{v} \cdot (x_0, \dots, x_{n-1}) \mid \mathbf{v} + \mathbf{u} \in C, \mathbf{u} \in \sum_{x_i \geq 1} C_{q_i}\}$. By this, we substract the C_{q_i} if the counter for q_i is greater or equal to one, that is, the state has been visited. This finishes the construction.

We now prove that $\mathcal{A}'$ is equivalent to $\mathcal{A}$. In the one direction we compress the run by using the appropriate shortcuts, in the other direction we unravel it accordingly.

To show that $SR_\omega(\mathcal{A}) \subseteq SR_\omega(\mathcal{A}')$, let $\alpha \in SR_\omega(\mathcal{A})$ with accepting run $r = r_1 r_2 r_3 \dots$. If there are no ε-transitions in r, we are done (as r is also an accepting run of $\mathcal{A}'$ on α).

Otherwise, we construct an accepting run r' of $\mathcal{A}'$ on α by replacing maximal ε-sequences in r step-by-step. Let i be minimal such that $r_i \dots r_j$ is a maximal ε-sequence. Let $r_i = (p_{i-1}, \varepsilon, \mathbf{v}_i, p_i), r_j = (p_{j-1}, \varepsilon, \mathbf{v}_j, p_j)$, and $r_{j+1} = (p_j, \alpha_z, \mathbf{v}_{j+1}, p_{j+1})$. It might be the case that $i = 1$, i. e., the run r starts with an ε-transition leaving q_0. Otherwise $i > 1$ and we can write $r_{i-1} = (p_{i-2}, \alpha_{z-1}, \mathbf{v}_{i-1}, p_{i-1})$. By allowing the empty sequence, we may assume that there is always a second (possibly empty) maximal ε-sequence $r_{j+2} \dots r_k$ starting directly after r_{j+1}. We distinguish (the combination of) the following cases.

- At least one state in $r_i \dots r_j$ is accepting, i. e., there is a position $i - 1 \leq \ell \leq j$ such that $p_\ell \in F$ (F) or not (N).
- At least one state in $r_{j+2} \dots r_k$ is accepting, i. e., there is a position $j + 1 \leq \ell' \leq k$ such that $p_{\ell'} \in F$ (F) or not (N). If $r_{j+2} \dots r_k$ is empty, we are in the case (N).

Hence, we consider four cases in total.

- Case (NN). That is, there is no accepting state in $r_i \dots r_k$. Note that the ε-sequence $r_i \dots r_j$ can be decomposed into an ε-path and ε-cycles as follows. If we have $p_{i_1} \neq p_{j_1}$ for all $i \leq i_1 < j_1 \leq j$ we are done as $r_i \dots r_j$ is already an ε-path. Otherwise let $i_1 \geq i$ be minimal such that there is $j_1 > i_1$ with $p_{i_1} = p_{j_1}$, that is, $r_{i_1+1} \dots r_{j_1}$ is an ε-cycle. If $r_i \dots r_{i_1} r_{j_1+1} \dots r_j$ is an ε-path, we are done. Otherwise, let $i_2 > j_1$ be minimal such that there is $j_2 > i_2$ with $p_{i_2} = p_{j_2}$, that is, $r_{i_2+1} \dots r_{j_2}$ is an ε-cycle. Then again, if $r_i \dots r_{i_1} r_{j_1+1} \dots r_{i_2} r_{j_2+1} \dots r_j$ is an ε-path, we are done. Otherwise, we can iterate this

argument and obtain a set of ε-cycles $r_{i_1+1} \dots r_{j_1}, \dots, r_{i_m+1} \dots r_{j_m}$ for some m, and an ε-path $\hat{r}_{i,j} = r_i \dots r_{i_1} r_{j_1+1} \dots r_{i_m} r_{j_m+1} \dots r_j$ which partition $r_i \dots r_j$. Now observe that $\rho(r_{i_1+1} \dots r_{j_1}) + \dots + \rho(r_{i_m+1} \dots r_{j_m}) \in C_{p_{i_1}} + \dots + C_{p_{i_m}}$. We can do the same decomposition for the ε-sequence $r_{j+2} \dots r_k$ into a set of ε-cycles and an ε-path $\hat{r}_{j+2,k}$. By the construction of Δ', there is a shortcut

$$\delta = (p_{i-1}, \alpha_z, (\rho(\hat{r}_{i,j}) + \mathbf{v}_{j+1} + \rho(\hat{r}_{j+2,k})) \cdot \hat{\mathbf{e}}^n, p_k),$$

where $\hat{\mathbf{e}}^n$ is the n-dimensional vector counting the states appearing in $\hat{r}_{i,j}$ and $\hat{r}_{j+2,k}$ and the state p_{j+1}. By the construction of Δ' and C', we may subtract all ε-cycles that have been visited in $r_i \dots r_k$, hence, we may replace $r_i \dots r_k$ by δ to simulate exactly the behavior of $\mathcal{A}$.

- Case (NF). That is, there is no accepting state in $r_i \dots r_j$ but at least one accepting state in $r_{i+2} \dots r_k$ (in particular, this sequence is not empty). Let $\ell_1, \dots, \ell_m$ denote the positions of accepting states in $r_{i+2} \dots r_k$, and let $\ell_0 < \ell_1$ be maximal such that ℓ_0 is resetting (this is before r_i, and if such an ℓ_0 does not exist, let $\ell_0 = 0$), i. e., ℓ_0 is the position of the last reset before the reset at position ℓ_1. As r is an accepting run, the sequence $S = (\ell_1, \dots, \ell_m)$ is a C-sequence (we may assume that all states in S are pairwise distinct, otherwise there is a reset-cycle, which can be ignored). In the same way as in the previous case we can partition the ε-sequence $r_i \dots r_j$ into an ε-path $\hat{r}_{i,j}$ and a set of ε-cycles, which may be subtracted from C. Likewise, we can partition the sequence $r_{j+2} \dots r_{\ell_1}$ into an ε-path $\hat{r}_{j+2,\ell_1}$ and ε-cycles with the same property. By the construction of Δ' there is a shortcut $(p_{i-1}, a, \rho(\hat{r}_{i,j}) + \mathbf{v}_{j+1}, p_{j+1})$ and hence a transition
$$\delta = (p_{i-1}, a, \rho(\hat{r}_{i,j}) + \mathbf{v}_{j+1} + \rho(\hat{r}_{j+2,\ell_1}), (p_{j+1}, S, p_k)).$$
Note that this is also the case if $i = 1$. Thus, we replace $r_i \dots r_k$ by δ. In particular, $\rho(r_{\ell_0+1} \dots r_{i-1}\delta)$ can be obtained from $\rho(r_{\ell_0+1} \dots r_{\ell_1})$ by subtracting all ε-cycles that have been visited within this partial run. Furthermore, observe that the containment of $\rho(r_{\ell_1+1} \dots r_{\ell_2}), \dots, \rho(r_{\ell_{m-1}+1} \dots r_{\ell_m})$ in C depends only on the automaton, and not the input word. As the counters are reset in r_{ℓ_m}, we may continue the run from δ the same way as in r_k, using an appropriate transition from Δ' that adds the vector $\rho(\hat{r}_{\ell_m+1,k})$, thus respecting the acceptance condition.
- Case (FN). Similar to (NF), but this time we replace $r_{i-1} r_i \dots r_j$ by an appropriate transition into a state of the form $(p_{i-2}, \alpha_z, \mathbf{v}, (p_{i-1}, S, p_j))$ for a suitable C-sequence S and vector $\mathbf{v}$, followed by a shortcut leading to p_k. If $i = 0$ (we enter a C-sequence before reading the first symbol), we make use of the transitions introduced especially for q_0.
- Case (FF). Similar to (FN) and (NF), but we transition from a state of the form (p_{i-1}, S, p_j) into a state of the form (p_{j+1}, S', p_k) for suitable C-sequences S, S', again respecting the case $i = 0$.

To show that $SR_\omega(\mathcal{A}') \subseteq SR_\omega(\mathcal{A})$ we unravel the shortcuts and (p, S, q)-states introduced in the construction. Let $\alpha \in SR_\omega(\mathcal{A}')$ with accepting run $r' = r'_1 r'_2 r'_3 \dots$. We replace every transition $r'_i \in \Delta' \setminus \Delta$ (i. e., transitions that do not appear in $\mathcal{A}$) by an appropriate sequence of transitions in $\mathcal{A}$. Let $i \geq 1$ be minimal such that r'_i is a transition in $\Delta' \setminus \Delta$.

We distinguish the form of r'_i and show that the possible forms correspond one-to-one to the cases in the forward direction.

- Case (NN). The case that $r'_i = (p, a, \rho(\tilde{\lambda}), q)$ is a shortcut, i. e., $\tilde{\lambda} \in \mathcal{E}^* \Delta \mathcal{E}^*$, corresponds to the case (NN). In particular, there are no accepting states in r. Let $k < i$ be the position of the last reset before r'_i, and k' the position of the first reset after r'_i, where $k' = i$ if r'_i transitions into a accepting state. By the acceptance condition we have $\rho(r'_{k+1} \dots r'_{k'}) \in C - (\sum_{q \in Q'} C_q)$ for some set $Q' \subseteq Q$ based on the counter values. Hence, we can replace r'_i by the partial run $\tilde{\lambda}$ filled with possible ε-cycles on some states in Q'.
- Case (NF). The case that $r'_i = (s, a, \mathbf{v} + \rho(\lambda), (p, S, q))$ such that $S = (f_1, \dots f_\ell)$ is a C-sequence, there is a transition $\delta = (s, a, \mathbf{v}, p) \in \Delta$ and λ is a no-reset ε-path from p to f_1, corresponds to the case (NF). By the definition of C-sequence there is a sequence r_{f_1, f_ℓ} of ε-transitions in $\mathcal{A}$ starting in f_1, ending in f_ℓ, visiting the accepting states f_1 to f_ℓ (in that order) such that the reset-acceptance condition is satisfied on every visit of one the accepting states. Then we can replace r'_i by $\delta \lambda r_{f_1, f_\ell}$, possibly again filled with some ε-cycles based on the state counters of λ, similar to the previous case. Note that at this point we do not yet unravel the path from f_ℓ to q, as it depends on how the run r' continues (as handled by the next two cases).
- Case (FN). The case that $r'_i = ((p, S, q), a, \mathbf{v} + \rho(\lambda), t)$ such that $S = (f_1, \dots f_\ell)$ is a C-sequence, there is a transition $\delta = (q, a, \mathbf{v}, t) \in \Delta$ and λ is a no-reset ε-path from f_ℓ to q, corresponds to the case (FN). Similar to the previous case, we can replace r'_i by $\lambda\delta$, possibly again amended with some ε-cycles based on the state counters of λ. If $i = 1$, the transition might also be of the form $r'_1 = (q_0, \alpha_1, \rho(\lambda) + \mathbf{v}, t)$ such that S is a C-sequence with the property that (q_0, f_1) is a C-pair. Then there is a sequence of ε-transitions r_{q_0, f_ℓ} in $\mathcal{A}$ as above. Then we replace r'_1 by $r_{q_0, f_\ell} \lambda \delta$ (with possible ε-cycles) instead.
- Case (FF). The case that $r'_i = ((p, S, q), a, \rho(\tilde{\lambda}), (p', S', q')$ such that $S = (f_1, \dots f_\ell)$ and $S' = (f'_1, \dots, f'_k)$ are C-sequences, there is a transition $\delta = (q, a, \mathbf{v}, p') \in \Delta$ and $\tilde{\lambda} = \lambda \delta \lambda'$, where λ is a no-reset ε-path from f_ℓ to q and λ' is a no-reset ε-path from p' to f'_1, corresponds to the case (FF). This case can be seen as the union of the previous cases. There is a sequence $r_{f'_1, f'_k}$ of ε-transitions in $\mathcal{A}$, as in the case (RF). Hence, we replace r'_i by $\tilde{\lambda} r_{f'_1, f'_k}$ (with possible ε-cycles). If $i = 1$, the transition might also be of the form $r'_1 = (q_0, \alpha_1, \rho(\lambda) + \mathbf{v} + \rho(\lambda'), (p', S', q'))$ such that (q_0, f_1) is a C-pair. Then there is a sequence of ε-transitions r_{q_0, f_ℓ} in $\mathcal{A}$ as above, and we replace r'_1 by $r_{q_0, f_\ell} \tilde{\lambda} r_{f'_1, f'_k}$.

Observe that the size of $\mathcal{A}'$ is in $O(|\mathcal{A}|^2 |\mathcal{A}|!)$. This finishes the proof of the lemma. ◀

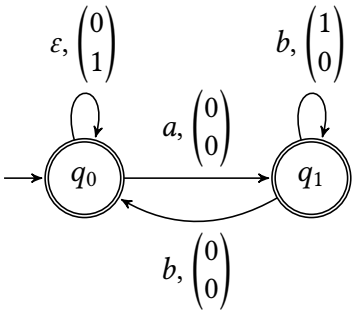


Figure 3.3. The ε-PA with $C = \{(z, z') \mid z' \geq z\}$ for the proof of Lemma 3.1.26.

Finally we show that safety and co-Büchi PA do not admit ε-elimination.

Lemma 3.1.26. *ε-safety PA and ε-co-Büchi PA do not admit ε-elimination.*

Proof. Consider the automaton $\mathcal{A}$ in Figure 3.3 with $C = \{(z, z') \mid z' \geq z\}$.

If we interpret $\mathcal{A}$ as an ε-safety or ε-co-Büchi PA, we have we have $S_\omega(\mathcal{A}) = CB_\omega(\mathcal{A}) = \{ab^+\}^\omega$. This ω-language is neither safety PA nor co-Büchi PA recognizable (one can easily adapt the proof in [GJLZ22, Theorem 3] showing that $\{\alpha \in \{a, b\}^\omega \mid |\alpha|_a = \infty\}$ is neither safety PA nor co-Büchi PA recognizable).

Observe how $\mathcal{A}$ utilizes the ε-transition to enforce that q_0 is seen infinitely often: whenever the b-loop on q_1 is used, the first counter increments. The semi-linear set states that at no point the first counter value may be greater than the second counter value which can only be increased using the ε-loop on q_0. Hence, any infinite word accepted by $\mathcal{A}$ may contain arbitrary infixes of the form b^n for $n < \infty$, as the automaton can use the ε-loop on q_0 at least n times before, but not b^ω. ◀

We generalize the trick presented in the previous proof to show that ε-co-Büchi PA recognize all Büchi PA recognizable ω-languages. Further adapting the trick we show that ε-safety PA recognize all reset PA recognizable ω-languages.

Lemma 3.1.27. *The class of Büchi PA recognizable ω-languages is a strict subclass of the class of ε-co-Büchi PA recognizable ω-languages.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a Büchi PA of dimension d. Without loss of generality we assume $q_0 \notin F$ (this can be achieved by adding a fresh initial state). We construct an equivalent ε-co-Büchi PA $\mathcal{A}'$ of dimension $d+3$ as follows. Let $Q' = Q \cup \{q_0'\} \cup \{q_f' \mid q_f \in F\}$ where q_0' is a fresh state and the q_f' are copies of the accepting states of $\mathcal{A}$ in which we do not expect good counter values. We define $\mathcal{A}' = (Q', \Sigma, q_0', \Delta', \mathcal{E}', Q', C')$. The idea is as follows: we use one of the three new counters to indicate that we would like to check the

current counter values for membership in C. The other two additional counters are used to enforce that we see a state in F infinitely often, that is that we indeed must check the current counter values for membership in C infinitely often. To achieve that, we introduce ε-loops on every state that is accepting in $\mathcal{A}$ as well as on the new initial state. The automaton then guesses the number of symbols that are read before visiting the next accepting state and increases the second counter accordingly. The first counter is incremented every time a symbol of the infinite input word is read. At every state we expect the second counter value to be greater or equal the first counter value.

Hence, we define

$$\begin{aligned}\Delta' = &\{(p, a, \mathbf{v} \cdot (1,0,0), q) \mid (p, a, \mathbf{v}, q) \in \Delta, p, q \notin F\}\\ &\cup \{(p, a, \mathbf{v} \cdot (1,0,1), q_f), (p, a, \mathbf{v} \cdot (1,0,0), q'_f) \mid (p, a, \mathbf{v}, q_f) \in \Delta, p \notin F, q_f \in F\}\\ &\cup \{(p_f, a, \mathbf{v} \cdot (1,0,1), q), (p'_f, a, \mathbf{v} \cdot (1,0,0), q) \mid (p_f, a, \mathbf{v}, q) \in \Delta, p_f \in F, q \notin F\}\\ &\cup \{(p'_f, a, \mathbf{v} \cdot (1,0,1), q_f), (p_f, a, \mathbf{v} \cdot (1,0,1), q'_f) \mid (p_f, a, \mathbf{v}, q_f) \in \Delta, p_f, q_f \in F\}\\ &\cup \{(p_f, a, \mathbf{v} \cdot (1,0,0), q_f), (p'_f, a, \mathbf{v} \cdot (1,0,0), q'_f) \mid (p_f, a, \mathbf{v}, q_f) \in \Delta, p_f, q_f \in F\},\end{aligned}$$

and

$$\mathcal{E}' = \{(q'_0, \varepsilon, \mathbf{0}^d \cdot (0,0,1), q_0), (q'_0, \varepsilon, \mathbf{0}^d \cdot (0,1,0), q'_0)\} \cup \{(q_f, \varepsilon, \mathbf{0}^d \cdot (0,1,0), q_f) \mid q_f \in F\}.$$

Finally, we define

$$\begin{aligned}C' = &\{\mathbf{0}^d \cdot (0, y, 0) \mid y \in \mathbb{N}\}\\ &\cup \{(v_1, \ldots, v_d, x, y, 2p+1) \mid p \in \mathbb{N}, y \geq x, (v_1, \ldots, v_d) \in \mathbb{N}^d\}\\ &\cup \{(v_1, \ldots, v_d, x, y, 2p+2) \mid p \in \mathbb{N}, y \geq x, (v_1, \ldots, v_d) \in C\}.\end{aligned}$$

The interested reader can now (fairly) easy verify that $\mathcal{A}$ and $\mathcal{A}'$ are equivalent.

The strictness of the inclusion follows immediately from [GJLZ22], as there are ω-languages recognized by co-Büchi PA but not by any Büchi PA. ◀

Before we show that ε-safety PA are powerful enough to simulate reset PA, we first introduce the following Myhill-Nerode-like notion for linear sets. Let C be a linear set of dimension d. We call C *congruent* if for every $\mathbf{v}_1, \mathbf{v}_2 \in C$ and every $\mathbf{v} \in \mathbb{N}^d$ we have $\mathbf{v}_1 + \mathbf{v} \in C$ if and only of $\mathbf{v}_2 + \mathbf{v} \in C$. We call a congruent linear set C *resetting* if $\mathbf{0} \in C$ (i. e., if C is congruent and homogeneous).

Lemma 3.1.28. *The class of reset PA recognizable ω-languages is a strict subclass of the class of ε-safety PA recognizable ω-languages.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a strong reset PA of dimension d with $C = \bigcup_{i \leq \ell} C(\mathbf{b}_i, P_i)$. We assume again without loss of generality that $q_0 \notin F$. We construct an equivalent ε-safety PA $\mathcal{A} = (Q', \Sigma, q_0', \Delta', \mathcal{E}', Q', C')$ of dimension $2d + 3$. The idea is to combine the ideas from the previous proof while observing that we can simulate resets by testing membership in a resetting linear set. To be precise, the additional three counters serve the same purpose as in the last proof while the additional d counters allow us to simulate resets. Hence, we start by defining $C' = C_{\mathsf{check}} \cup C_{\mathsf{don't\ care}}$ where

$$C_{\mathsf{check}} = \{(v_1, \ldots, v_d, v_1, \ldots, v_d, z, z, 2p) \mid p, z, v_1, \ldots, v_d \in \mathbb{N}\}$$

and

$$C_{\mathsf{don't\ care}} = \{(v_1, \ldots, v_{2d} x, y, 2p + 1) \mid p \in \mathbb{N}, y \geq x, (v_1, \ldots, v_{2d}) \in \mathbb{N}^{2d}\}.$$

The crucial observation is that C_{check} is a resetting linear set. In the next step we introduce a new initial state that is equipped with an ε-loop and a ε-transition to q_0 exactly as the in previous proof. Then we replace every accepting state $f \in F$ by the gadget depicted in Figure 3.4. Here we use the following observation: whenever a vector $\mathbf{v}$ is contained in C, then it is in particular contained $C(\mathbf{b}_i, P_i)$ for a $i \leq \ell$ and can hence be written as $\mathbf{b}_i + \sum_{\mathbf{p} \in P_i} \mathbf{p} z_{\mathbf{p}}$ for some values $z_{\mathbf{p}} \in \mathbb{N}$. While the first d counters of $\mathcal{A}'$ are copies of the d counters of $\mathcal{A}$, the second set of d counters are used to represent any vector that is contained in C, i. e., can be written as mentioned above. The ε-loops in the state $f^{(i)}$ allow us to add any vector contained in $C(\mathbf{b}_i, P_i)$ to the counter values of the second set of d counters. Hence, the vector induced by the first d counters is contained in C if and only if f can be reached with a vector whose vector induced by the second set of d counters is equivalent to the vector induced by the first d counters. This is checked in the state f where membership in C_{check} is tested (as the last counter has an even value when entering f).

We refrain from giving the details of the construction of $\mathcal{A}'$ as they do not yield further insights. ◀

Observe that ε-safety PA generalize ε-co-Büchi PA, as they can guess the number n of transitions a ε-co-Büchi PA uses before they constantly verify the Parikh-condition. Very similar to the previous proof, the ε-safety PA uses an ε-transition on a fresh initial state n times to postpone the verification of the Parikh-condition for n transitions.

We conclude by arguing that ε-co-Büchi PA do not generalize reset PA. To achieve that, we define the following property, which is heavily inspired by the pumping-style lemma for Parikh recognizable (finite word) languages, see Lemma 2.2.3.

Let $\mathcal{L}$ be a class of ω-languages. We say that $\mathcal{L}$ has the *exchange property* if for every $L \in \mathcal{L}$ there are $p, \ell \in \mathbb{N}$ such that for every $\alpha \in L$ the following holds: there is a partition $\alpha = uvxv\beta$ with $0 < |v| \leq p < |x|$ and $|uvxv| \leq \ell$ such that $uv^2x\beta \in L$ and $uxv^2\beta \in L$.

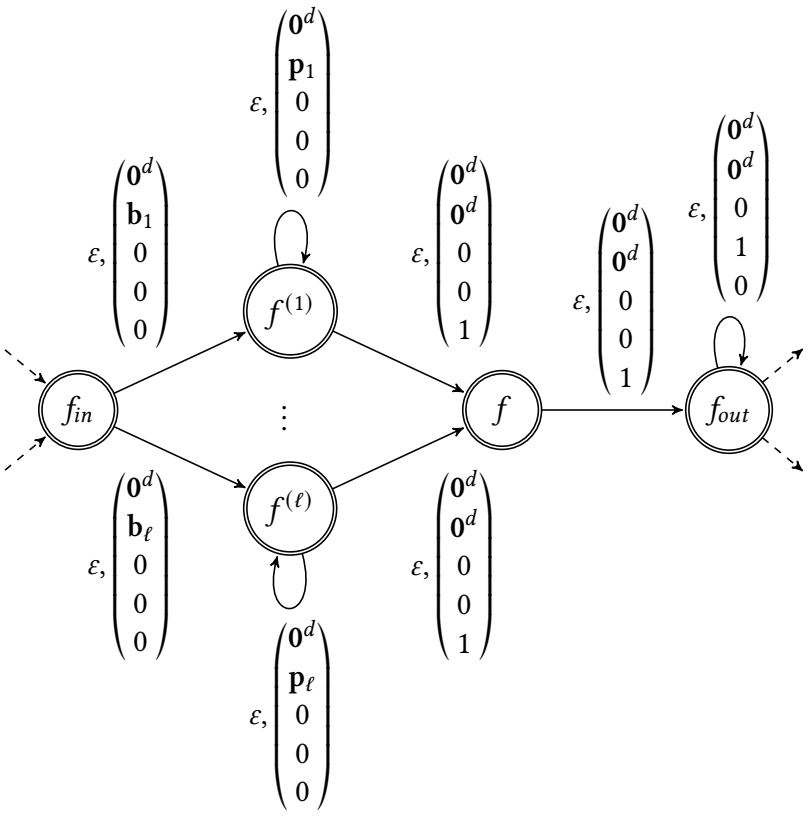


Figure 3.4. We replace every accepting state $f \in F$ by such a gadget. Every state $f^{(i)}$ is equipped with an ε-loop for every $\mathbf{p}_i \in P_i$. Note that the actual membership of the current counter values in C is (still) checked in f as we require the current counter values to be in C_{check}.

Observation 3.1.29. *The classes of Büchi PA and (ε-)co-Büchi PA recognizable ω-languages have the exchange property. The classes of reset PA and (ε-)safety PA recognizable ω-languages do not have the exchange property.*

As an immediate consequence of this observation and the argument above we obtain the following corollaries.

Corollary 3.1.30. *The class of ε-co-Büchi PA recognizable ω-languages is a strict subset of the class of ε-safety PA recognizable ω-languages.*

Corollary 3.1.31. *The classes of reset PA recognizable ω-languages and ε-co-Büchi PA recognizable ω-languages are incomparable.*

3.1.8 Remaining Closure Properties

As a preparation for the next subsection we establish the remaining closure properties. As limit PA and reachability-regular PA are equivalent by Theorem 3.1.8, we only argue for limit PA. Similarly, as strong reset PA and weak reset PA are equivalent by Lemma 3.1.23 and Lemma 3.1.24, we only argue for strong reset PA.

First we observe that the ω-languages recognized by reset PA are ultimately periodic.

Lemma 3.1.32. *Let $\mathcal{A}$ be a reset PA. If $SR_\omega(\mathcal{A}) \neq \emptyset$, then $\mathcal{A}$ accepts an infinite word of the form uv^ω.*

Proof. Assume $SR_\omega(\mathcal{A}) \neq \emptyset$. Then there exists an infinite word $\alpha \in SR_\omega(\mathcal{A})$ with accepting run $r = r_1 r_2 r_3 \ldots$, where $r_i = (p_{i-1}, \alpha_i, \mathbf{v}_i, p_i)$. Let $k_1 < k_2 < \ldots$ be the positions of all accepting states in r. Let $k_i < k_j$ be two such positions such that $p_{k_i} = p_{k_j}$. Let $u = \alpha_1 \ldots \alpha_{k_i}$ be the prefix of α read upon visiting p_{k_i} and $v = \alpha_{k_i+1} \ldots \alpha_{k_j}$ the infix read between p_{k_i} and p_{k_j}. Then $\mathcal{A}$ also accepts uv^ω, as $r_1 \ldots r_{k_i}(r_{k_i+1} \ldots r_{k_j})^\omega$ is an accepting run of $\mathcal{A}$ on uv^ω by definition. ◀

Lemma 3.1.33. *The classes of limit PA recognizable and reachability-regular PA recognizable ω-languages are closed intersection. The class of reset PA recognizable ω-languages is not closed under intersection.*

Proof. The closure of limit PA (and hence reachability-regular PA) under intersection follows from a simple product construction, exactly as for finite word PA [KR03b, Theorem 21]. The non-closure of reset PA under intersection follows from an argument similar to the argument showing non-closure of blind counter machines [FS08] (on infinite words). Let $L_1 = \{a^n b^n \mid n > 0\}^\omega$ and $L_2 = \{a\}\{b^n a^{2n} \mid n > 0\}$. Then $L_1 \cap L_2$ contains only one infinite word, namely $aba^2b^2a^4b^4 \ldots$ which is not ultimately periodic. Hence, $L_1 \cap L_2$ is not accepted by any strong reset PA as a consequence of the previous lemma. ◀

Finally, we show that all our models are not closed under complement.

Lemma 3.1.34. *The classes of limit PA recognizable, reachability-regular PA recognizable, and reset PA recognizable ω-languages are not closed complement.*

Proof. Non-closure for reset PA follows immediately from the closure under union and non-closure under intersection by De Morgan's law.

Hence, we only need to argue for limit PA. Let $D = \{ww \mid w \in \{a,b\}^*\}$. As shown in [KR03b, Lemma 26], this language is not Parikh recognizable. However, its complement $\overline{D}$ is Parikh recognizable. Now let $L = \overline{D} \cdot \{c\}^\omega$. By Lemma 3.1.2, this ω-language is limit PA recognizable. Observe that $\overline{L} = \{a,b\}^\omega \cup \{a,b\}^*\{c\}^*\{a,b\}\{a,b,c\}^\omega \cup PAL \cdot \{c\}^\omega$. The first two languages of the union are ω-regular and hence limit PA recognizable. Hence, it is sufficient to show that $D \cdot \{c\}^\omega$ is not limit PA recognizable. Observe that limit PA have the exchange property. Cadilhac et al. [CFM11] have shown how to establish the pumping-syle lemma (Lemma 2.2.3) to show that the language $\{w\#w \mid w \in \{a,b\}^*\}$ is not Parikh recognizable. The proof can easily adapted to show that D is not Parikh recognizable (one only needs to remove the #), which again can easily adapted to show that $D \cdot \{c\}^\omega$ and hence $\overline{L}$ is not recognized by any limit PA by exploiting the exchange property. ◀

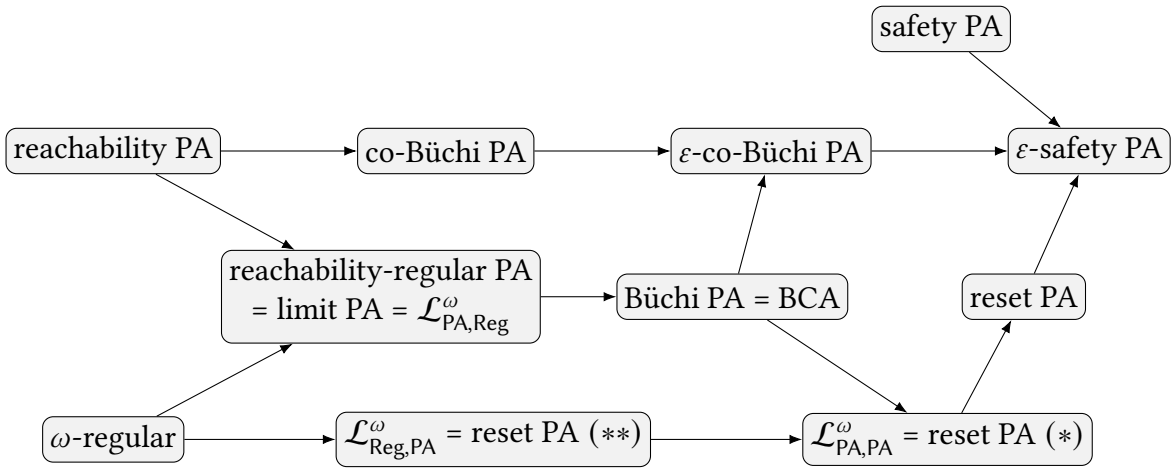
At this point we are ready to present a complete picture of comparing all (non-deterministic) models, see Figure 3.5.

3.1.9 Decision Problems

In this section, we study the following classical decision problems for PA on infinite words.

- Emptiness: given a PA $\mathcal{A}$, is the ω-language of $\mathcal{A}$ empty?
- Membership: given a PA $\mathcal{A}$ and finite words u, v, does $\mathcal{A}$ accept uv^ω?
- Universality: given a PA $\mathcal{A}$, does $\mathcal{A}$ accept every infinite word?

Furthermore, we study the classical model checking problem, where we are given a system $\mathcal{K}$ and a specification, e. g., represented as an automaton $\mathcal{A}$, and the question is whether every run of $\mathcal{K}$ satisfies the specification, i. e., we ask $L(\mathcal{K}) \subseteq L(\mathcal{A})$, which is true if and only if $L(\mathcal{K}) \cap \overline{L(\mathcal{A})} = \emptyset$. However, as complementing is often expensive or not even possible, another approach is to specify the set of all bad runs and ask whether no run of $\mathcal{K}$ is bad, which boils down to the question is $L(\mathcal{K}) \cap L(\mathcal{A}) = \emptyset$? We call the first approach *universal model checking* and the latter approach *existential model checking*. In our setting we assume the specification $\mathcal{A}$ to be a PA operating on infinite words, while the system $\mathcal{K}$ may be given as a Kripke-structure (which can be seen as a safety automaton [CHVB18]), in which case the goal is to solve *safety model checking*, or also as a PA operating on infinite words,



(*) At most one state q per leaf of $C(\mathcal{A})$ may have incoming transitions from outside the leaf, this state q is the only accepting state in the leaf, and there are no accepting states in non-leaves;
(**) and only transitions connecting states in leaves may be labeled with non-zero vectors.

Figure 3.5. Comparison of Parikh automata on infinite words. Arrows indicate strict inclusions while no (non-transitive) connections mean incomparability.

in which case the goal is to solve *PA model checking*. Hence, we consider four problems in total, which boil down to the following decision problems.

- Inclusion: given a safety automaton or a PA $\mathcal{A}_1$, and a PA $\mathcal{A}_2$, is the ω-language of $\mathcal{A}_1$ a subset of the ω-language of $\mathcal{A}_2$?
- Intersection emptiness: given a safety automaton or a PA $\mathcal{A}_1$, and a PA $\mathcal{A}_2$, is the ω-language of $\mathcal{A}_1$ disjoint from the ω-language of $\mathcal{A}_2$?

Observe that our characterization results imply that we can efficiently translate reachability-regular PA into Büchi PA which in turn can efficiently be translated into strong reset PA. Furthermore strong reset PA and weak reset PA can efficiently be translated into each other by Lemma 3.1.23 and Lemma 3.1.24. Hence, for showing upper bounds for these models it is sufficient to argue for reset PA. However, the proof of Lemma 3.1.12 showing that every limit PA recognizable ω-language is a member of $\mathcal{L}^{\omega}_{\text{PA,Reg}}$ constructs a number of finite word PA that is exponentially in the dimension of the limit PA we start with. Hence, we argue for limit PA separately. Contrary, we can translate reachability-regular PA efficiently into limit PA; hence, for showing lower bounds of all newly introduced models it is sufficient to argue for reachability-regular PA.

We begin by showing that emptiness for reset PA is coNP-complete, where the upper bound even holds if we allow ε-transitions (which does not increase their expressiveness but our

ε-elimination procedure constructs an equivalent reset PA of super-polynomial size). Hence, reset PA (even with ε-transitions) are a powerful model that can still be used for algorithmic applications.

We present an algorithm solving non-emptiness for reset PA in NP by exploiting that ω-languages accepted by reset PA are ultimately periodic. As a consequence, we can reduce non-emptiness for ε-reset PA to the finite word case, as clarified in the following lemma.

Lemma 3.1.35. *Emptiness for reachability-regular PA, strong reset PA and weak reset PA is* coNP*-complete.*

Proof. The coNP-hardness for reachability-regular PA (and hence all of these models) follows immediately from the coNP-hardness of finite word PA [FL15]. Hence, we focus on the membership in coNP by presenting an NP-algorithm for non-emptiness that works even for strong reset PA with ε-transitions. Let $\mathcal{A}$ be a strong ε-reset PA. By Lemma 3.1.32 it suffices to check whether $\mathcal{A}$ accepts an ultimately periodic infinite word uv^ω. If such a word exists, we may assume that there is an accepting run $r_u r_v^\omega$ of $\mathcal{A}$ on uv where neither r_u nor r_v visit the same accepting state twice (otherwise we simply remove such cycles in the run). For any $p, q \in Q$ we define $\mathcal{A}_{p\Rightarrow q} = (Q \cup \{q_0'\}, \Sigma, q_0', \Delta', \mathcal{E}', \{q\}, C)$, where $\Delta' = \{(q_1, a, \mathbf{v}, q_2) \mid (q_1, a, \mathbf{v}, q_2) \in \Delta, q_1 \notin F\} \cup \{(q_0', a, \mathbf{v}, q_2) \mid (p, a, \mathbf{v}, q_2) \in \Delta\}$ and, analogously, $\mathcal{E}' = \{(q_1, \varepsilon, \mathbf{v}, q_2) \mid (q_1, \varepsilon, \mathbf{v}, q_2) \in \mathcal{E}, q_1 \notin F\} \cup \{(q_0', \varepsilon, \mathbf{v}, q_2) \mid (p, \varepsilon, \mathbf{v}, q_2) \in \Delta\}$.

Hence, if we interpret $\mathcal{A}_{p\Rightarrow q}$ as a finite word PA, then it accepts all words accepted by the finite word PA $\mathcal{A}$ when starting in p, ending in q, and not visiting an accepting state in-between. In other words, if $p, q \in F$, then $\mathcal{A}_{p\Rightarrow q}$ accepts all finite infixes that the strong reset PA $\mathcal{A}$ may read when the last reset was in p, and the next reset is in q

Now, the following NP algorithm solves non-emptiness:

1. Guess a sequence $f_1, \ldots, f_k$ of accepting states with $k \leq 2|F|$ such that $f_i = f_k$ for some $i \leq k$.
2. Verify that $L(\mathcal{A}_{q_0\Rightarrow f_1}) \neq \emptyset$ and $L(\mathcal{A}_{f_j\Rightarrow f_{j+1}}) \neq \emptyset$ for all $1 \leq j < k$ (interpreted as PA over finite words).
3. Verify that $L(\mathcal{A}_{f_i\Rightarrow f_{i+1}}) \cdot \ldots \cdot L(\mathcal{A}_{f_{k-1}\Rightarrow f_k}) \not\subseteq \{\varepsilon\}$.

The third step can be done by adding a fresh symbol (say e) to the automata and replacing every ε-transition with an e-transition (observe that this does construction does not change the emptiness behavior, and is, in contrast to the ε-elimination procedure in [KR03b] computable in polynomial time). Afterwards we use the NP-algorithm for non-emptiness for PA [FL15].

The third step essentially states that not all $L(\mathcal{A}_{f_j\Rightarrow f_{j+1}})$ for $j \geq i$ may only accept the empty word, as we require $v \neq \varepsilon$. To check this property, we can construct a PA[1] recognizing

[1]This is possible in polynomial time by a standard construction very similar to the one of Lemma 3.1.2.

$L(\mathcal{A}_{f_i \Rightarrow f_{i+1}}) \cdot \ldots \cdot L(\mathcal{A}_{f_{k-1} \Rightarrow f_k})$, and again replace every ε-transition with an e-transition. Finally, we build the product automaton with the PA (NFA) that recognizes the language $\{w \in (\Sigma \cup \{e\})^* \mid w \text{ contains at least 1 symbol from } \Sigma\}$, which is possible in polynomial time [KR03b] and test non-emptiness for the resulting PA. ◀

Let us quickly explain how to modify the algorithm such that we obtain an NP-algorithm for non-emptiness for limit PA.

Lemma 3.1.36. *Emptiness for limit PA is* coNP*-complete.*

Proof. Let $\mathcal{A}$ be a limit PA of dimension d. Recall the proof of Lemma 3.1.12 showing $L_\omega(\mathcal{A})$ is a member of $\mathcal{L}^\omega_{\text{PA,Reg}}$ by iterating over all subsets $D \subseteq [d]$. Intuitively, such a subset D indicates the counters which we expect to be ∞ when processing an infinite word $\alpha \in L_\omega(\mathcal{A})$. For every fixed D, we utilize Lemma 3.1.11 to compute a sequence of finite word PA, say $\mathcal{A}_1, \mathcal{A}'_1, \ldots \mathcal{A}_n, \mathcal{A}'_n$ for some $n \geq 1$ in polynomial time such that $\alpha \in \bigcup_{i \leq n} L(\mathcal{A}_i) \cdot L(\mathcal{A}'_i)^\omega = L$. Given these automata, we can compute a reachability-regular PA recognizing L in polynomial time by Corollary 3.1.10. In particular, for every fixed D we can compute a reachability-regular PA (and hence a reset PA by the comment above) accepting all words that are accepted by $\mathcal{A}$ such that there is an accepting run r of $\mathcal{A}$ on α with $\mathsf{Inf}(\rho(r)) = D$ (recall that $\mathsf{Inf}(\mathbf{v})$ for some $\mathbf{v} \in \mathbb{N}^d$ denotes the positions of all ∞-entries in $\mathbf{v}$). Hence, we can use the NP-algorithm in the previous proof by first guessing a good set D and turning only the matching "relevant part" of $\mathcal{A}$ into a reset PA. ◀

We will now turn our attention to the membership problem. Note that, given finite words u and v, we can always construct a safety automaton that recognizes uv^ω and no other infinite word with $|uv|$ many states. Recall that every state of a safety automaton is accepting. We show that the intersection of a reset PA recognizable ω-language and a safety automaton-recognizable ω-language remains reset PA recognizable using a product construction which is computable in polynomial time. Hence, we can reduce the membership problem to the non-emptiness the standard way.

Lemma 3.1.37. *The class of reset PA recognizable ω-languages is closed under intersection with safety automata-recognizable ω-languages.*

Proof. We show a construction for strong ε-reset PA that is computable in polynomial time. Let $\mathcal{A}_1 = (Q_1, \Sigma, q_1, \Delta_1, \mathcal{E}_1, F_1, C_1)$ be a strong ε-reset PA and $\mathcal{A}_2 = (Q_2, \Sigma, q_2, \Delta_2, Q_2)$ be a safety automaton. Consider the product automaton

$$\mathcal{A} = (Q_1 \times Q_2, \Sigma, (q_1, q_2), \Delta, \mathcal{E}, F_1 \times Q_2, C_1)$$

with

$$\Delta = \{((p, q), a, \mathbf{v}, (p', q') \mid (p, a, \mathbf{v}, p') \in \Delta_1 \text{ and } (q, a, q') \in \Delta_2\}$$

and

$$\mathcal{E} = \{((p, q), \varepsilon, \mathbf{v}, (p', q)) \mid (p, \varepsilon, \mathbf{v}, p') \in \mathcal{E}_1 \text{ and } q \in Q_2\}.$$

As every state of $\mathcal{A}_2$ is accepting, we need to take care that $\mathcal{A}$ does not use a transition that is not enabled in $\mathcal{A}_2$ while mimicking the behavior of $\mathcal{A}_1$. Hence, it is easily verified that $SR_\omega(\mathcal{A}) = SR_\omega(\mathcal{A}_1) \cap L_\omega(\mathcal{A}_2)$. ◀

Combining the previous two results, we obtain that membership for all our models is in NP. For the lower bound, we reduce from the membership problem for semi-linear sets, that is, given a semi-linear set $C \subseteq \mathbb{N}^d$ and a vector $\mathbf{v} \in \mathbb{N}^d$, deciding membership of $\mathbf{v}$ in C. This problem is known to be NP-complete, which follows immediately from the NP-algorithms for integer programming [BT76, vzGS78] and the NP-hardness of (a variant of) subset sum [GJ79, Kar72]; see also [Haa18] for a short discussion.

Corollary 3.1.38. *Membership for limit PA, reachability-regular PA, strong reset PA and weak reset PA is* NP*-complete.*

We will now turn our attention to the universality and inclusions problems, the latter being the core of solving universal model checking. Note that we can always reduce universality to inclusion, as an automaton $\mathcal{A}$ is universal if and only if Σ^ω is a subset of the ω-language of $\mathcal{A}$. Observe however that universality (and hence inclusion) remain undecidable for our models, as these problems are already undecidable for reachability PA [GJLZ22] which can effectively be translated into reachability-regular PA. This implies that the universal model checking problems are undecidable for our models.

Corollary 3.1.39. *Universality and inclusion are undecidable for limit PA, reachability-regular PA, strong reset PA and weak reset PA.*

Contrary, the existential safety model checking for all our models, that is, intersection emptiness for PA with safety automata is coNP-complete. Hardness follows immediately from the hardness of emptiness, while containment in coNP is as an immediate consequence of Lemma 3.1.37.

Corollary 3.1.40. *Intersection-emptiness for limit PA, reachability-regular PA, strong reset PA and weak reset PA with safety automata is* coNP*-complete.*

We continue with the existential PA model checking problem. This problems is coNP complete for limit PA and reachability-regular PA. Again, hardness follows from the hardness of emptiness, while containment follows from the closure under intersection, as witnessed by the product construction in Lemma 3.1.33 which is computable in polynomial time

Lemma 3.1.41. *Intersection-emptiness for limit PA and reachability-regular PA is* coNP*-complete.*

We continue with Büchi PA, as their intersection-emptiness problems has not been studied in the literature before. Their intersection-emptiness problem remains coNP-complete; however, we need more sophisticated methods to proof this statement.

Lemma 3.1.42. *Intersection-emptiness for Büchi PA is* coNP*-complete.*

Proof. The lower bound follows again from their coNP-complete emptiness problem.

We give a proof sketch showing that intersection non-emptiness for Büchi PA is in NP by utilizing a recent result essentially stating that Ramsey-quantifiers in Presburger formulas can be eliminated in polynomial time [BGLZ24]. The authors show how to use the Ramsey-quantifier to check liveness properties for systems with counters. In particular, the existence of an accepting run of a Büchi PA (answering the question whether the accepted ω-language is non-empty) can be expressed with a Presburger formula with a Ramsey-quantifier. Hence, checking if the intersection of the two ω-languages recognized by two Büchi PA can be tested by intersecting two Presburger-formulas and moving the quantifiers to the front. We refer to Sections 4.1 and 8.2 in [BGLZ24] for more information. ◀

We conclude by showing that intersection emptiness is undecidable for strong reset PA. The result relies on the fact that the intersection of two such languages can encode non-terminating computations of two-counter machines.

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

$$(1 : \mathsf{I}_1)(2 : \mathsf{I}_2) \dots (k-1 : \mathsf{I}_{k-1})(k : \mathsf{STOP})$$

where the first component of a pair (ℓ, I_ℓ) is the line number, and the second component is the instruction in line ℓ. An instruction is of one of the following forms:

- $\mathsf{Inc}(Z_i)$, where $i = 0$ or $i = 1$.
- $\mathsf{Dec}(Z_i)$, where $i = 0$ or $i = 1$.
- If $Z_i = 0$ goto ℓ' else ℓ'', where $i = 0$ or $i = 1$, and $\ell', \ell'' \leq k$.

Instructions of the first or second form are called increments resp. decrements, while instructions of the latter form are called zero-tests. A configuration of $\mathcal{M}$ is a tuple $c = (\ell, z_0, z_1)$, where $\ell \leq k$ is the current line number, and $z_0, z_1 \in \mathbb{N}$ are the current counter values of Z_0 and Z_1 respectively. We say c *derives* into its unique successor configuration c', written $c \vdash c'$, as follows.

- If $\mathsf{I}_\ell = \mathsf{Inc}(Z_0)$, then $c' = (\ell + 1, z_0 + 1, z_1)$.
- If $\mathsf{I}_\ell = \mathsf{Inc}(Z_1)$, then $c' = (\ell + 1, z_0, z_1 + 1)$.
- If $\mathsf{I}_\ell = \mathsf{Dec}(Z_0)$, then $c' = (\ell + 1, \max\{z_0 - 1, 0\}, z_1)$.
- If $\mathsf{I}_\ell = \mathsf{Dec}(Z_1)$, then $c' = (\ell + 1, z_0, \max\{z_1 - 1, 0\})$.

- If $\mathsf{I}_\ell = \mathsf{If}\ Z_0 = 0$ goto ℓ' else ℓ'', then $c' = (\ell', z_0, z_1)$ if $z_0 = 0$, and $c' = (\ell'', z_0, z_1)$ if $z_0 > 0$.
- If $\mathsf{I}_\ell = \mathsf{If}\ Z_1 = 0$ goto ℓ' else ℓ'', then $c' = (\ell', z_0, z_1)$ if $z_1 = 0$, and $c' = (\ell'', z_0, z_1)$ if $z_1 > 0$.
- If $\mathsf{I}_\ell = \mathsf{STOP}$, then c has no successor configuration.

The unique computation of $\mathcal{M}$ is a finite or infinite sequence of configurations $c_0c_1c_2\ldots$ such that $c_0 = (1, 0, 0)$ and $c_i \vdash c_{i+1}$ for all $i \geq 0$. Observe that the computation is finite if and only if the instruction $(k : \mathsf{STOP})$ is reached. If this is the case, we say $\mathcal{M}$ terminates. Given a two-counter machine $\mathcal{M}$, it is undecidable to decide whether $\mathcal{M}$ terminates [Min67].

In the following we assume without loss of generality that our two-counter machines satisfy the guarded-decrement property [GJLZ22], which guarantees that every decrement does indeed change a counter value: every decrement $(\ell : \mathsf{Dec}(Z_i))$ is preceded by a zero-test of the form $(\ell - 1, \mathsf{If}\ Z_i = 0$ goto $\ell + 1$ else $\ell)$. Note that this modification does not change the termination behavior of a two-counter machine, as decrementing a counter whose value is already zero does not have an effect.

Lemma 3.1.43. *The intersection emptiness problem for reset PA is undecidable.*

Proof. We can encode infinite computations of two-counter machines as infinite words over $\Sigma = \{a, b, 1, 2, \ldots, k\} \cup \Sigma_\mathsf{I}$, where $\Sigma_\mathsf{I} = \{I_a, I_b, D_a, D_b, Z_a, Z_b, \bar{Z}_a, \bar{Z}_b\}$. The idea is as follows. Let $c = (\ell, z_0, z_1)$ be a configuration of $\mathcal{M}$. We encode c as a finite word $w_c = \ell ux \in \Sigma^*$, where $\ell \in \{1, 2, \ldots, k\}$ encodes the current line number, $u \in \{a, b\}^*$ with $|u|_a = z_0$ and $|u|_b = z_1$ encodes the current counter values, and $x \in \Sigma_\mathsf{I}$ encodes the instruction I_ℓ of line ℓ as follows:

- If $\mathsf{I}_\ell = \mathsf{Inc}(Z_0)$, then $x = I_a$, and if $\mathsf{I}_\ell = \mathsf{Inc}(Z_1)$, then $x = I_b$.
- If $\mathsf{I}_\ell = \mathsf{Dec}(Z_0)$, then $x = D_a$, and if $\mathsf{I}_\ell = \mathsf{Dec}(Z_1)$, then $x = D_b$.
- If $\mathsf{I}_\ell = \mathsf{If}\ Z_0 = 0$ goto ℓ' else ℓ'', and the line number of the unique successor configuration of c is ℓ', then $x = Z_a$ (that is, the zero-test is successful). Analogously with $x = Z_b$.
- If $\mathsf{I}_\ell = \mathsf{If}\ Z_0 = 0$ goto ℓ' else ℓ'', and the line number of the unique successor configuration of c is ℓ'', then $x = \bar{Z}_a$ (that is, the zero-test fails). Analogously with $x = \bar{Z}_b$.

Let $w_c, w_{c'} \in \Sigma^*$ be two words encoding two configurations of $\mathcal{M}$. We call $w_c \cdot w_{c'}$ *correct* if $c \vdash c'$. Hence, we can encode a unique infinite computations $c_0c_1c_2\ldots$ as an infinite word $w_{c_0}w_{c_1}w_{c_2}\ldots$. We show that the ω-language $L = \{w_{c_0}w_{c_1}w_{c_2}\ldots\}$ can be written as the intersection of two deterministic strong reset PA ω-languages. Let

$$L_1 = \{w_{c_0}w_{c_1}w_{c_2}\cdots \mid w_{c_{2i}}w_{c_{2i+1}} \text{ is correct for every } i \geq 0\}, \text{ and}$$
$$L_2 = \{w_{c_0}w_{c_1}w_{c_2}\cdots \mid w_{c_{2i+1}}w_{c_{2i+2}} \text{ is correct for every } i \geq 0\}.$$

Observe that $L_1 \cap L_2 = L$, and L is empty if and only if the unique computation of $\mathcal{M}$ terminates. Hence, it remains to show that L_1 and L_2 are recognized by deterministic strong reset PA. We argue for L_1; the argument for L_2 is very similar. The idea is as follows: We construct a deterministic strong reset PA $\mathcal{A}_1$ with five counters that tests the correctness of two consecutive encodings of configurations, say $w_{c_{2i}} \cdot w_{c_{2i+1}} = \ell_1 u_1 x_1 \cdot \ell_2 u_2 x_2$ with $\ell_1, \ell_2 \in \{1, 2, \ldots, k\}$, $u_1 u_2 \in \{a, b\}^*$ and $x_1, x_2 \in \Sigma_{\mathsf{I}}$. First observe that checking whether ℓ_2 is indeed the correct line number (that is, the correct successor of ℓ_1) can be hard-coded into the state space of $\mathcal{A}_1$: if x_1 encodes an increment or decrement, we expect $\ell_2 = \ell_1 + 1$, and if x_1 encodes a successful or failing zero-test If $Z_i = 0$ goto ℓ' else ℓ'', we expect $\ell_2 = \ell'$ or $\ell_2 = \ell''$, respectively. Four counters of $\mathcal{A}_1$ are used to count the numbers of a's and b's in u_1 and u_2, respectively. Then, if $x_1 = I_a$, we expect $|u_2|_a = |u_1|_a + 1$ and $|u_2|_b = |u_1|_b$, and so on. To be able to perform the correct check, we also encode x_1 into the state space as well as the fifth counter by counting modulo $|\Sigma_{\mathsf{I}}|$. Observe that the guarded-decrement property ensures that decrements are handled correctly. Hence, $\mathcal{A}_1$ has two sets of states counting the numbers of a's and b's of u_1 and u_2, accordingly, as well as a set of accepting states that is used to check the counter values.

After such a check, $\mathcal{A}_1$ resets, and continues with the next two (encodings of) configurations. The automaton for L_2 works in the same way, but skips the first configuration. ◀

3.2 Deterministic Parikh automata

In this section, we study the deterministic variants of Parikh automata on infinite words. First, we study the closure properties of the newly introduced models, yielding the foundation in order to investigate the expressiveness of the models. The main part of this section is devoted to the decision problems; in particular we focus on the core problems for model checking that, in contrast to the non-deterministic variants, are decidable for the deterministic variants and present algorithms for these problems.

3.2.1 Closure Properties

We now study the closure properties of the deterministic variants of the models introduced by Grobler et al. [GSS24], that is, deterministic limit PA, deterministic reachability-regular PA, deterministic strong reset PA, and deterministic weak reset PA.

It is well known that semi-linear sets over $\mathbb{N}^d$ are closed under complement [GS64]; see also [Haa18]. Before we study deterministic limit automata we show that this is also true for semi-linear sets enriched with ∞.

Lemma 3.2.1. *Let $C \subseteq \mathbb{N}_\infty^d$ be a semi-linear set. Then the complement $\bar{C} = \mathbb{N}_\infty^d \setminus C$ is semi-linear.*

Proof. Let $f : \mathbb{N}_\infty \to \mathbb{N}$ be the bijection with $f(\infty) = 0$ and $f(i) = i + 1$ for $i \in \mathbb{N}$. We extend f to vectors $\mathbf{v} = (v_1, \ldots, v_d) \in \mathbb{N}_\infty^d$ and sets of vectors $C \subseteq \mathbb{N}_\infty^d$ component-wise: $f(v_1, \ldots, v_d) = (f(v_1), \ldots, f(v_d))$ and $f(C) = \{f(\mathbf{v}) \mid \mathbf{v} \in C\}$. Note that $f(C) \subseteq \mathbb{N}^d$.

Now we observe that $f(C)$ is semi-linear if and only if C is semi-linear. First assume that C is semi-linear. We may assume that $C = C(\mathbf{b}, P)$ is linear, as we can carry out the following procedure for every linear set individually. Assume $\mathbf{b} = (b_1, \ldots, b_d)$. We define $D_\infty(\mathbf{b}) = \{i \mid b_i = \infty\}$. For a set $D \subseteq \{1, \ldots, d\}$ with $D_\infty(\mathbf{b}) \subseteq D$ and a vector $\mathbf{v} = (v_1, \ldots, v_d) \in \mathbb{N}_\infty^d$, let $\mathbf{v}^D = (v_1^D, \ldots, v_d^D)$ with $v_i^D = 0$ if $i \in D$ and $v_i^D = v_i$ if $i \notin D$. Furthermore, we call a subset $P' \subseteq P$ of period vectors *D-compatible* if for every $i \in D \setminus D_\infty(\mathbf{b})$, the set P' contains at least one vector where the ith component is ∞, and if for every $i \notin D$, the set P' contains no vector where the ith component is ∞. Observe that this definition ensures that for every vector $\mathbf{v} \in P'$ we have $\mathbf{v}^D \in \mathbb{N}^d$, that is, no component in $\mathbf{v}^D$ is ∞. Let $\mathcal{P}_\infty^D \subseteq 2^P$ be the collection of D-compatible subsets of P. Then we have

$$f(C) = \bigcup_{D_\infty(\mathbf{b}) \subseteq D \subseteq \{1,\ldots,d\}} \bigcup_{P' \in \mathcal{P}_\infty^D} C(\mathbf{b}^D + \mathbf{1}^D + \sum_{\mathbf{p} \in P'} \mathbf{p}^D, \{\mathbf{p}^D \mid \mathbf{p} \in P'\}),$$

which is semi-linear by definition.

For the other direction, we may again assume that $f(C) = C(\mathbf{b}, P)$ is linear. Similar to above, assume $\mathbf{b} = (b_1, \ldots, b_d)$ and define $D_0(\mathbf{b}) = \{i \mid b_i = 0\}$. For a set $D \subseteq D_0(\mathbf{b})$ we call a

subset $P' \subseteq P$ of period vectors *D-safe* if for every $i \in D$ the set P' contains no vector where the ith component is not 0, and if for every $i \notin D$ the set P' contains at least one vector where ith component is not 0. Let $\mathcal{P}_0^D \subseteq 2^P$ be the collection of D-safe subsets of P. Let $\mathbf{i}_D = (v_1, \ldots, v_d)$ with $v_i = \infty$ if $i \in D$ and $v_i = 0$ if $i \notin D$. Then we have

$$C = \bigcup_{D \subseteq D_0(\mathbf{b})} \bigcup_{P' \in \mathcal{P}_0^D} C(\mathbf{i}_D + \mathbf{b} - \mathbf{1} + \sum_{\mathbf{p} \in P'} \mathbf{p}, P'),$$

which is semi-linear by definition (we assume $\infty - 1 = \infty$). Observe that every component in $\mathbf{i}_D + \mathbf{b} + \sum_{\mathbf{p} \in P'} \mathbf{p}$ is strictly greater 0, hence we can subtract $\mathbf{1}$ without getting negative.

As semi-linear sets over $\mathbb{N}^d$ are closed under complement, we have

$$\begin{aligned} C \text{ is semi-linear} &\Leftrightarrow f(C) \text{ is semi-linear} \\ &\Leftrightarrow \mathbb{N}^d \setminus f(C) \text{ is semi-linear} \\ &\Leftrightarrow f^{-1}(\mathbb{N}^d \setminus f(C)) = \overline{C} \text{ is semi-linear.} \end{aligned}$$ ◀

Lemma 3.2.2. *The class of deterministic limit PA recognizable languages is closed under union, intersection and complement.*

Proof. First observe that we can always assume that every state of a (deterministic) limit PA is accepting, as we can check the existence of an accepting state being visited infinitely often in the semi-linear set. To achieve this, we introduce one new counter and increment it at every transition that points to an accepting state. In the semi-linear set we enforce that this counter is ∞.

Hence, we can show the closure under union and intersection by a standard product construction. In case of union, we test whether at least one automaton has good counter values, and we can show the closure under intersection by testing whether both automata have good counter values. Closure under complement follows immediately from Lemma 3.2.1. ◀

Following the standard proof showing that the language $L_{a<\infty} = \{\alpha \in \{a, b\}^\omega \mid |\alpha|_a < \infty\}$ is not deterministic ω-regular, we make the following observation.

Observation 3.2.3. *There is no deterministic reachability-regular PA, deterministic weak reset PA or deterministic strong reset PA recognizing the ω-regular language $L_{a<\infty}$.*

Observe however, that the complement $L_{a=\infty} = \{\alpha \in \{a, b\}^\omega \mid |\alpha|_a = \infty\}$ of $L_{a<\infty}$ is recognized by all of these models.

Lemma 3.2.4. *The class of deterministic reachability-regular PA recognizable languages is not closed under union, intersection or complement.*

Proof. First we show non-closure under union. Let

$$L_1 = \{uc\alpha \mid u \in \{a,b,c\}^*, |u|_a = |u|_b, |\alpha|_c = \infty\}$$

and

$$L_2 = \{va\beta \mid v \in \{a,b,c\}^*, |v|_b = |v|_c, |\beta|_a = \infty\}.$$

Both languages are deterministic reachability-regular PA recognizable, as witnessed by the automaton in Figure 3.6 and the fact that L_2 can be obtained from L_1 by shuffling symbols. We show that the language $L_1 \cup L_2$ is not deterministic reachability-regular PA recognizable. Assume there is an n-state deterministic reachability-regular PA $\mathcal{A}$ recognizing $L_1 \cup L_2$. Then there is a unique accepting run $r_1 r_2 \dots$ of $\mathcal{A}$ on abc^ω. In particular, at some point the automaton verifies the Parikh condition, say after using the transition r_i. Let $m = \max\{n+1, i\}$ and consider the infinite word $abc^m b^{m-1} a^\omega$ with the unique accepting run $r'_1 r'_2 \dots$ of $\mathcal{A}$. Due to determinism, we have $r_j = r'_j$ for all $j \leq m+2$; in particular, the automaton verifies the Parikh condition within this run prefix. As $m - 1 \geq n$, the automaton visits a state, say q twice while reading b^{m-1}. Hence, we can pump this q-cycle and obtain an accepting run of $\mathcal{A}$ on $abc^m b^{m-1+k} a^\omega$ for some $k > 0$, a contradiction.

For the non-closure under intersection, define

$$L_1 = \{\alpha \mid |\alpha[1:i]|_a = |\alpha[1:i]|_b \text{ for some } i\}$$

and

$$L_2 = \{\alpha \mid |\alpha[1:i]|_a = |\alpha[1:i]|_c \text{ for some } i\}.$$

Suppose there is an n-state deterministic reachability-regular PA recognizing $L_1 \cap L_2$. Let $\alpha = a(a^n b^n)^{n+1} c^{n(n+1)+1} a^\omega$. The unique run r of $\mathcal{A}$ on α is not accepting, as α has no balanced a-b prefix. Observe that $\mathcal{A}$ visits at least one state twice while reading a b^n-block. Furthermore, there is a state, say q, such that $\mathcal{A}$ visits q twice while reading two different b^n-blocks, as there are $n+1$ different b^n-blocks. Hence, we can swap the latter q-cycle to the front and obtain an infinite word, say α' in $L_1 \cap L_2$, with a unique accepting run r' of $\mathcal{A}$. This run verifies the Parikh condition at some point. We distinguish two cases. If r' verifies the Parikh condition before reading the first c, we can depump the $c^{n(n+1)+1}$-block and obtain an accepting run on an infinite word without a balanced a-c-prefix, a contradiction. Hence assume that r' verifies the Parikh condition after reading at least one c, say at position k. However, then we have $\rho(r[1:k]) = \rho(r'[1:k])$, and hence $\mathcal{A}$ also accepts α, a contradiction.

The non-closure under complement follows immediately from Observation 3.2.3. ◀

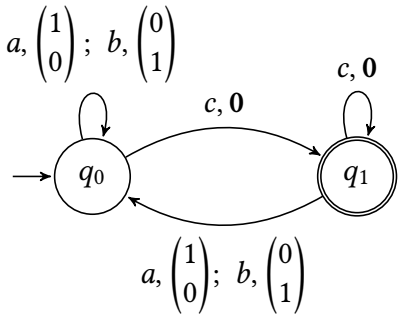


Figure 3.6. The deterministic reachability-regular PA with $C = C(\mathbf{0}, \{\mathbf{1}\})$ for $L_1 = \{uc\alpha \mid u \in \{a,b,c\}^*, |u|_a = |u|_b, |\alpha|_c = \infty\}$.

Lemma 3.2.5. *The class of deterministic weak reset PA recognizable languages is not closed under union, intersection or complement.*

Proof. We begin with the non-closure under union. Let

$$L_{a=b} = \{\alpha \mid |\alpha[1:i]|_a = |\alpha[1:i]|_c \text{ for } \infty \text{ many } i\}$$

and similarly define

$$L_{a=c} = \{\alpha \mid |\alpha[1:i]|_a = |\alpha[1:i]|_c \text{ for } \infty \text{ many } i.$$

We show that $L_{a=b} \cup L_{a=c}$ is not deterministic weak reset PA recognizable. Assume there is a deterministic weak reset PA $\mathcal{A}$ recognizing $L_{a=b} \cup L_{a=c}$. Consider the unique accepting run r of $\mathcal{A}$ on $\alpha = (ab)^\omega$ and let i, j be two positions with $i+1 < j$ such that r resets after reading $\alpha[1:i]$ and $\alpha[1:j]$ in the same state (such a pair of positions does always exists by the infinite pigeonhole principle). Now consider the infinite word $\alpha[1:i]c^{|\alpha[1:i]|_a}(ac)^\omega$, which is also accepted by $\mathcal{A}$. However, this implies that $\mathcal{A}$ also accepts $\alpha[1:j]c^{|\alpha[1:i]|_a}(ac)^\omega$, as $\mathcal{A}$ is in the same (accepting) state after reading $\alpha[1:i]$ as well as $\alpha[1:j]$, but this infinite word is not contained in $L_{a=b} \cup L_{a=c}$, as $\alpha[1:j]$ contains at least one more a than $\alpha[1:i]$, a contradiction.

The argument for the non-closure under intersection is the same as for the non-deterministic setting, as the ω-languages considered in Lemma 3.1.33 are indeed deterministic weak reset PA recognizable.

The non-closure under complement is an immediate consequence of Observation 3.2.3. ◀

Lemma 3.2.6. *The class of deterministic strong reset PA recognizable languages is not closed under union, intersection or complement.*

Proof. We begin with the non-closure under union. Let $L = \{c^*a^nc^*b^n \mid n > 0\}^\omega$ and $L_{c=\infty} = \{\alpha \mid |\alpha|_c = \infty\}$. Observe that $a^nc^\omega \in L \cup L_{c=\infty}$ for every $n \geq 0$. Assume there is a deterministic strong reset PA recognizing $L \cup L_{c=\infty}$. Let $n_1 \neq n_2$ be such that the unique accepting runs of $\mathcal{A}$ on $\alpha_1 = a^{n_1}c^\omega$ resp. $\alpha_2 = a^{n_2}c^\omega$ reset in the same state the first time they reset after reading at least one c, say after reading $\alpha_1[1 : i_1]$ resp. $\alpha_2[1 : i_2]$ (with $i_1 > n_1$ and $i_2 > n_2$). As $a^{n_1}c^{i_1-n_1}b^{n_1}(ab)^\omega$ is also accepted by $\mathcal{A}$, the infinite word $a^{n_2}c^{i_2-n_2}b^{n_1}(ab)^\omega$ is also accepted by $\mathcal{A}$, a contradiction.

To show the non-closure under intersection, we use an argument similar to the non-deterministic setting. Let $L_1 = \{a^nb^n \mid n > 0\}^\omega$ and $L_2 = \{a\}\{b^na^{2n} \mid n > 0\}$. Then $L_1 \cap L_2$ contains only one infinite word, namely $aba^2b^2a^4b^4 \ldots$. Hence $L_1 \cap L_2$ is not ultimately periodic and hence not accepted by any strong reset PA [GSS24].

The non-closure under complement again follows from Observation 3.2.3. ◀

3.2.2 Expressiveness

In this section we study the expressiveness of deterministic PA on infinite words for those models whose deterministic variants were not studied before in the literature.

First we remark that deterministic reachability-regular PA, deterministic limit PA, deterministic strong reset PA and deterministic weak reset PA are strictly weaker than their non-deterministic counterparts. This follows immediately from their different closure properties: reachability-regular PA, weak reset PA (and hence strong reset PA) are closed under union (see Lemma 3.1.1), and limit PA are not closed under complement (see Lemma 3.1.33). Hence, from the results of the previous section we obtain the following corollary.

Corollary 3.2.7. *The following strict inclusions hold.*

- *Deterministic reachability-regular PA ⊊ Reachability-regular PA.*
- *Deterministic limit PA ⊊ Limit PA.*
- *Deterministic strong reset PA ⊊ Reset PA.*
- *Deterministic weak reset PA ⊊ Reset PA.*

In the following, when we show non-inclusions, we always give the strongest separation, e. g., when showing that a deterministic strong reset PA cannot simulate a deterministic weak reset PA, we show that it can not even simulate a deterministic reachability PA, which is weaker than a deterministic weak reset PA. We refer to Figure 3.7 for an overview of the results in this section. We simply write that a model is a strict/no subset of another model; by that we mean that the class of ω-languages recognized by the first model is a strict/no subset of the class of ω-languages recognized by the second model.

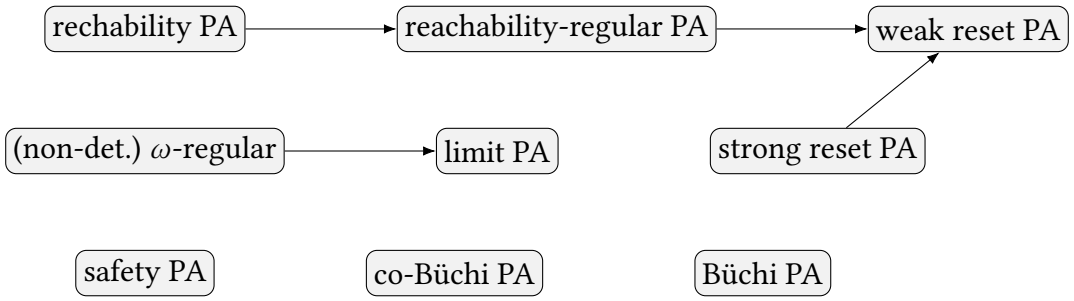


Figure 3.7. Inclusion diagram of the studied *deterministic* models. Arrows indicate strict inclusions while no connections mean incomparability.

ω-regular languages

We begin by showing that every ω-regular language is deterministic limit PA recognizable.

Lemma 3.2.8. *ω-regular $\subsetneq$ deterministic limit PA.*

Proof. Let $L \subseteq \Sigma^\omega$ be ω-regular and let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, \mathcal{F})$ be a deterministic Muller automaton recognizing L. The idea is to construct an equivalent deterministic limit PA $\mathcal{A}' = (Q, \Sigma, q_0, \Delta', Q, C)$ of dimension $|Q|$, where every state is accepting, while encoding the sets in $\mathcal{F}$ into the semi-linear set C. Let $f : Q \to \{1, \dots, |Q|\}$ be a bijection associating every state with a counter. Hence, we define $\Delta' = \{(p, a, \mathbf{e}_{f(q)}^{|Q|}, q) \mid (p, a, q) \in \Delta\}$. For every $F \in \mathcal{F}$, we define $C_F = C(\sum_{q \in F} \mathbf{i}_{f(q)}, \{\mathbf{e}_{f(q)} \mid q \notin F\})$. That is, for every state in F we expect its counter value to be ∞, while we expect every other counter value to be a finite number. We choose $C = \bigcup_{F \in \mathcal{F}} C_F$ and hence obtain an equivalent deterministic limit PA.

The strictness is witnessed by the ω-language $\{a^n b^n c^\omega \mid n > 0\}$, which is obviously deterministic limit PA recognizable, but not ω-regular. ◀

Observation 3.2.3 immediately yields the following result.

Corollary 3.2.9.

$$\omega\text{-regular} \not\subseteq \begin{cases} \text{Deterministic reachability-regular PA} \\ \text{Deterministic Büchi PA} \\ \text{Deterministic strong reset PA} \\ \text{Deterministic weak reset PA} \end{cases}$$

Observe however that these models generalize deterministic Büchi automata. This is not true for deterministic reachability PA, deterministic safety PA nor deterministic co-Büchi PA, as shown in the next lemma.

Lemma 3.2.10.

$$\textit{Deterministic } \omega\textit{-regular} \not\subseteq \begin{cases} \textit{Deterministic reachability PA} \\ \textit{Deterministic safety PA} \\ \textit{Deterministic co-Büchi PA} \end{cases}$$

Proof. As an immediate consequence of Lemma 3.2.12 (proved below) we have that no deterministic reachability PA recognizes the deterministic ω-regular language a^*b^ω.

The two other claims follow from [GJLZ22, Proof of Theorem 3], where the authors have shown that (even non-deterministic) safety PA do not recognize the deterministic ω-regular language $\{a,b\}^\omega \setminus \{a\}^\omega$ and that no co-Büchi PA recognizes the deterministic ω-regular language $L_{a=\infty} = \{\alpha \in \{a,b\}^\omega \mid |\alpha|_a = \infty\}$. ◀

Deterministic Safety PA and co-Büchi PA

As a consequence of Lemma 3.2.10 we obtain the following corollary.

Corollary 3.2.11.

$$\left.\begin{array}{l} \textit{Deterministic reachability-regular PA} \\ \textit{Deterministic limit PA} \\ \textit{Deterministic strong reset PA} \\ \textit{Deterministic weak reset PA} \end{array}\right\} \not\subseteq \begin{cases} \textit{Deterministic safety PA} \\ \textit{Deterministic co-Büchi PA} \end{cases}$$

As shown in [GJLZ22] also deterministic reachability PA $\not\subseteq$ deterministic safety PA and deterministic reachability PA $\not\subseteq$ deterministic co-Büchi PA as well as deterministic Büchi PA $\not\subseteq$ deterministic safety PA and deterministic Büchi PA $\not\subseteq$ deterministic co-Büchi PA. Furthermore, the classes of deterministic safety PA and deterministic co-Büchi PA are themselves incomparable as shown in [GJLZ22].

Vice versa, deterministic safety PA $\not\subseteq$ non-deterministic weak reset PA and deterministic co-Büchi PA $\not\subseteq$ non-deterministic weak reset PA [GSS24]. Hence, these classes are no subclasses of any of the other studied classes.

Overall, deterministic safety PA and deterministic co-Büchi PA are incomparable with all other studied models.

Deterministic Reachability PA

We begin by characterizing the class of deterministic reachability PA recognizable ω-languages.

Lemma 3.2.12. *An ω-language L is deterministic reachability PA recognizable if and only if $L = U\Sigma^\omega$, where $U \subseteq \Sigma^*$ is recognized by a deterministic PA.*

Proof. Let $\mathcal{A}$ be a deterministic reachability PA recognizing L. Then we have $L(\mathcal{A}) = U$. Likewise, if $\mathcal{A}$ is a PA recognizing U, then $R_\omega(\mathcal{A}) = L$ (recall that $\mathcal{A}$ is complete by the definition of determinism). ◀

We have the following strict inclusion.

Lemma 3.2.13. *Deterministic reachability PA $\subsetneq$ deterministic reachability-regular PA.*

Proof. Let $\mathcal{A}$ be a deterministic reachability PA. We may assume that every state of $\mathcal{A}$ is accepting, as we can project the current state into the semi-linear set. To be precise, we introduce two new counters for each state of $\mathcal{A}$, counting the number of visits and exits. Then, the current state is the (unique) state with one more visit than exit, or in case that the number of visits and exits is the same for every state, then the current state is the initial state of $\mathcal{A}$. As these statements can be encoded into a semi-linear set, we can assume that every state is equipped with its own semi-linear set, and can hence make every state accepting (and assign the empty set if we want to simulate a non-accepting state). If every state is accepting, then $\mathcal{A}$ is an equivalent deterministic reachability-regular PA.

The strictness is witnessed e. g., by the ω-language $\{a^n b^n a^\omega \mid n > 0\}$, which is deterministic reachability-regular PA recognizable and by Lemma 3.2.12 not deterministic reachability PA recognizable. ◀

It remains to show the following incomparability results.

Lemma 3.2.14. *Deterministic reachability PA $\not\subseteq$ deterministic limit PA.*

Proof. We show that the deterministic reachability PA recognizable ω-language

$$L = \{\alpha \mid |\alpha[1:i]|_a = |\alpha[1:i]|_b \text{ for some } i > 0\}$$

is not deterministic limit PA recognizable. The proof is similar [GJLZ22, Lemma 3]. Assume there is an n-state deterministic limit PA $\mathcal{A}$ recognizing L. Consider the unique non-accepting run of $\mathcal{A}$ on $a(a^n b^n)^\omega$. Observe that $\mathcal{A}$ visits at least one state twice while reading a b^n-block, and there are at least two of the (infinitely many) b^n-blocks such that $\mathcal{A}$ visits the same state, say q, twice while reading them. Hence, we can shift one such q-cycle to the front and obtain the unique run on an infinite word that is in L. However, this run is still non-accepting, as the extended Parikh image and number of visits of an accepting state do not change. ◀

Lemma 3.2.15. *Deterministic reachability PA $\not\subseteq$ deterministic strong reset PA.*

Proof. We show that the deterministic reachability PA recognizable ω-language

$$L = \{a^n b^n \mid n \geq 1\} \cdot \{a, b\}^\omega$$

is not deterministic strong reset PA recognizable. Assume there is a deterministic strong reset PA $\mathcal{A}$ with n states recognizing L. Let $\alpha = a^n b^\omega$ with unique accepting run $r = r_1 r_2 r_3 \dots$ of $\mathcal{A}$ on α. Let $f_1, f_2, \dots$ be the sequence of reset positions of r and let $i > n$ be minimal with $i = f_{i'}$ for some $i' \geq 1$ (that is, $f_{i'}$ is the first reset position after reading a b).

First observe that $i < 2n$. Assume that this is not the case. As $\mathcal{A}$ visits at least one state twice while reading b^n, say state q, we observe that $\mathcal{A}$ is caught in a $q \dots q$ cycle while reading b^ω due to determinism. That is, every state that is visited while reading b^ω is already visited while reading the first n many bs. Hence we have $i < 2n$. Now let $j \geq 2n$ be minimal such that $j = f_{j'}$ for some $j' > i'$ is a reset position in r such that the state at position $f_{j'}$ is the same state as the one at position $f_{i'}$ (which exists by the same argument).

Now let $\alpha' = a^n b^{j-n} a^\omega$ with unique accepting run $r' = r'_1 r'_2 r'_3 \dots$ of $\mathcal{A}$ on α'. Observe that $\alpha[1:j] = \alpha'[1:j]$, and hence $r[1:j] = r'[1:j]$. As the partial runs $r[1:i]$ and $r[1:j]$ reachability the same accepting state, the run $r[1:i] r'_{j+1} r'_{j+2} \dots$ is an accepting run of $\mathcal{A}$ on $a^n b^{i-n} a^\omega$. However, as $i - n < n$, this infinite word is not contained in L, a contradiction. ◀

Deterministic Reachability-regular PA

We begin by showing that every deterministic reachability-regular PA (and hence every deterministic reachability PA) can be translated into an equivalent deterministic weak reset PA.

Lemma 3.2.16. *Deterministic reachability-regular PA $\subsetneq$ deterministic weak reset PA.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a deterministic reachability-regular PA and let $\mathcal{A}' = (Q \cup \{q'_0\}, \Sigma, q'_0, \Delta', F, C')$ be a copy of $\mathcal{A}$ with a new fresh initial state q'_0 inheriting all outgoing transitions of q_0 (observe that this modification preserves determinism). We add one new counter that is incremented at every transition leaving q'_0, and not modified otherwise, that is,

$$\Delta' = \{(p, a, \mathbf{v} \cdot 0, q) \mid (p, a, \mathbf{v}, q) \in \Delta\} \cup \{(q'_0, a, \mathbf{v} \cdot 1, q) \mid (q_0, a, \mathbf{v}, q) \in \Delta\}.$$

We choose $C' = C \cdot \{1\} \cup \mathbb{N}^d \cdot \{0\}$ and obtain an equivalent weak reset PA $\mathcal{A}'$.

The strictness is witnessed by the ω-language $\{a^n b^n \mid n > 0\}^\omega$, which is obviously deterministic weak reset PA recognizable, but not even recognized by (non-deterministic) Büchi PA [GJLZ22], which are more expressive than reachability-regular PA. ◀

Deterministic Strong Reset PA

Lemma 3.2.17. *Deterministic strong reset PA $\subsetneq$ deterministic weak reset PA.*

Proof. The inclusion follows from Lemma 3.1.23, as the construction preserves determinism. The strictness follows from the fact that $\{a^n b^n \mid n \geq 1\} \cdot \{a, b\}^\omega$ is deterministic reachabil-

ity PA recognizable, and hence deterministic weak reset PA recognizable (by Lemma 3.2.13 and Lemma 3.2.16), but not recognized by any deterministic strong reset PA, as shown in Lemma 3.2.15. ◀

Lemma 3.2.18. *Deterministic strong reset PA $\not\subseteq$ deterministic Büchi PA.*

Proof. The argument is as in Lemma 3.2.16. The ω-language $\{a^n b^n \mid n > 0\}^\omega$ is deterministic strong reset PA recognizable, but there is no Büchi PA recognizing it [GJLZ22]. ◀

Lemma 3.2.19. *Deterministic strong reset PA $\not\subseteq$ deterministic limit PA.*

Proof. This follows from the previous proof as limit PA are less expressive than Büchi PA. ◀

Deterministic Büchi PA

We show that ω-languages recognized by deterministic Büchi PA can be characterized in a similar way as deterministic ω-regular languages.

Lemma 3.2.20. *An ω-language L is deterministic Büchi PA recognizable if and only of $L = \vec{P}$ where P is recognized by a deterministic PA.*

Proof. Let $\mathcal{A}$ be a deterministic Büchi PA recognizing L and let $\alpha \in B_\omega(\mathcal{A})$ with accepting run r. As r has infinitely many accepting hits by definition, we have $\alpha \in L(\vec{\mathcal{A}})$. Similarly, let $\mathcal{A}$ be a deterministic PA recognizing P and let $\alpha \in \vec{P}$. As $\mathcal{A}$ is deterministic, the unique run of $\mathcal{A}$ on α has infinitely many accepting hits, hence we have $\alpha \in B_\omega(\mathcal{A})$. ◀

Lemma 3.2.21. *Deterministic Büchi PA $\not\subseteq$ deterministic limit PA.*

Proof. The proof is almost identical to the proof of Lemma 3.2.14, but this time we consider the ω-language $L_{a=b} = \{\alpha \mid |\alpha[1:i]|_a = |\alpha[1:i]|_b \text{ for } \infty \text{ many } i\}$. Then we can re-use the same argument as the constructed infinite word has indeed infinitely many balanced a-b prefixes. ◀

Lemma 3.2.22. *Deterministic Büchi PA $\not\subseteq$ deterministic weak reset PA.*

Proof. As shown in Lemma 3.2.5, the ω-language $L_{a=b} \cup L_{a=c}$ is not deterministic weak reset PA recognizable. However, it is obviously deterministic Büchi PA recognizable. ◀

We note however that the class of ω-languages recognized by deterministic Büchi PA with a *linear* set form a subclass of the class of ω-languages recognized by deterministic weak reset PA with a linear set, as clarified in the following lemma.

Lemma 3.2.23. *Let $\mathcal{A}$ be a deterministic Büchi PA with a* linear set $C(\mathbf{b}, P)$. *Then there is an equivalent deterministic weak reset PA.*

Proof. First we observe that if $\mathbf{b} = \mathbf{0}$, then we have $B_\omega(\mathcal{A}) = WR_\omega(\mathcal{A})$. To see this, let $\alpha \in B_\omega(\mathcal{A})$ with (unique) accepting run r. By Dickson's Lemma [Dic13], the run r contains an infinite monotone sequence $s_1 < s_2 < \dots$ of accepting hits, that is, for all $i \geq 0$ we have $\rho(r[1 : s_i]) \in C(\mathbf{b}, P)$ and for all $j > i$ we have $\rho(r[s_i + 1 : s_j]) \in C(\mathbf{b}, P)$. Hence, the run r also satisfies the weak reset condition. For the other direction let $\alpha \in WR_\omega(\mathcal{A})$ with (unique) accepting run r and reset positions $0 = k_0 < k_1 < k_2 \dots$. As we assume $\mathbf{b} = \mathbf{0}$, it is immediate that $\rho(r[1 : k_i]) \in C(\mathbf{b}, P)$ for all $i \geq 1$. Hence, the run r also satisfies the Büchi condition.

Finally we argue that we can always assume that $\mathbf{b} = \mathbf{0}$. Indeed, we can always encode $\mathbf{b}$ into the state space of $\mathcal{A}$. ◀

3.2.3 Decision Problems and Model Checking

In this section, we study classical decision problems as well as the core problems for model checking for the deterministic variants. We repeat the problems for convenience. For an overview of the results in this section we refer to Table 1.2 and Table 1.3.

- Emptiness: given a PA $\mathcal{A}$, is the ω-language of $\mathcal{A}$ empty?
- Membership: given a PA $\mathcal{A}$ and finite words u, v, does $\mathcal{A}$ accept uv^ω?
- Universality: given a PA $\mathcal{A}$, does $\mathcal{A}$ accept every infinite word?
- Inclusion: given a safety automaton or a PA $\mathcal{A}_1$, and a PA $\mathcal{A}_2$, is the ω-language of $\mathcal{A}_1$ a subset of the ω-language of $\mathcal{A}_2$?
- Intersection-emptiness: given a safety automaton or a PA $\mathcal{A}_1$, and a PA $\mathcal{A}_2$, is the ω-language of $\mathcal{A}_1$ disjoint from the ω-language of $\mathcal{A}_2$?

The techniques employed for showing NP-completeness for non-emptiness and membership are identical to the non-deterministic case, see Lemma 3.1.35, Lemma 3.1.36 and Corollary 3.1.38.

Corollary 3.2.24. *Emptiness for deterministic limit PA, deterministic reachability-regular PA, deterministic weak reset PA and deterministic strong reset PA is* coNP*-complete.*

Corollary 3.2.25. *Membership for deterministic limit PA, deterministic reachability-regular PA, deterministic weak reset PA and deterministic strong reset PA is* NP*-complete.*

However, showing undecidability and completeness results for universality and the model checking problems require more sophisticated methods. Hence, we devote most of this section to the latter problems, being the core of solving universal model checking. Recall

that we can always reduce universality to inclusion, as an automaton $\mathcal{A}$ is universal if and only if Σ^ω is a subset of the ω-language of $\mathcal{A}$. Hence, we show all undecidability results and lower bounds for universality and all decidability results and upper bounds for inclusion. We begin with the undecidability results.

Lemma 3.2.26. *Universality for deterministic reachability-regular PA and deterministic weak reset PA is undecidable.*

Proof. As shown in [GJLZ22], universality is already undecidable for deterministic reachability PA. As we can effectively construct equivalent deterministic reachability-regular PA and deterministic weak reset PA from deterministic reachability PA by Lemma 3.2.13 and Lemma 3.2.16, the lemma follows. ◀

Recall our strategy for showing that universality for finite word PA is Π_2^{P}-hard by a reduction from the irrelevance problem for PA, see Corollary 2.4.4. We conclude Π_2^{P}-hardness for universality for deterministic limit and deterministic strong reset PA using a simplified variant of this reduction.

Corollary 3.2.27. *Universality and inclusion for deterministic limit PA and deterministic strong reset PA is Π_2^{P}-hard.*

Now we focus on the decidability results and upper bounds. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two (deterministic limit) PA. Note that $L_\omega(\mathcal{A}_1) \subseteq L_\omega(\mathcal{A}_2)$ holds if and only if $L_\omega(\mathcal{A}_1) \cap \overline{L_\omega(\mathcal{A}_2)} = \emptyset$. As deterministic limit PA are effectively closed under complement and intersection, and emptiness is coNP-complete (and hence decidable) for them, we obtain the following result.

Corollary 3.2.28. *Universality and inclusion for deterministic limit PA are decidable.*

Unfortunately, we do not obtain tight bounds and conjecture that these problem are Π_2^{P}-complete for them. However, we make the following observation. The relatively high Π_2^{P} lower bound of universality for deterministic limit PA and deterministic strong reset PA (and also of irrelevance for finite word PA) can be explained by the cost of (implicitly) complementing semi-linear sets. In fact, if we have the guarantee that the semi-linear set of the second PA can be complemented in polynomial time, the universality and inclusion problems become coNP-complete. We start with deterministic limit PA.

Lemma 3.2.29. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be deterministic limit PA with the guarantee that the semi-linear set of $\mathcal{A}_2$ can be complemented in polynomial time. Then the following questions are* coNP*-complete.*

1. *Is $L_\omega(\mathcal{A}_2) = \Sigma^\omega$?*
2. *Is $L_\omega(\mathcal{A}_1) \subseteq L_\omega(\mathcal{A}_2)$?*

Proof. Containment in coNP of both questions follows immediately from a reduction to emptiness for deterministic limit PA, as the guarantee allows us to complement deterministic limit PA in polynomial time, and we can construct the product automaton of two deterministic limit PA in polynomial time.

The partition problem is defined as follows: given a multiset M of positive integers, is there a subset M' of M such that $\sum_{n\in M} n = \sum_{n\in M\setminus M'} n$?. This problem is one of the classical NP-complete problems [GJ79]. We reduce from its complement.

Let $M = \{n_1, \dots n_k\}$ be a multiset of positive integers. We construct a deterministic limit PA $\mathcal{A}$ over the alphabet $\{a, b\}$ of dimension 2 as follows. The state set of $\mathcal{A}$ is $\{q_0, q_1, \dots q_k\}$ where q_0 is the initial state and q_k is the only accepting state. For every $1 \leq i \leq k$, there is an a-transition from q_{i-1} to q_i labeled with $(n_i, 0)$ as well as a b-transition labeled with $(0, n_i)$. Finally, we add an a-loop and a b-loop to the accepting state q_k, both labeled with $\mathbf{0}$. The semi-linear set of $\mathcal{A}$ is

$$C = \{(z, z') \mid z \neq z'\} = C((1,0), \{(1,0), (1,1)\}) \cup C((0,1), \{(0,1), (1,1)\}),$$

whose size does not depend on the size of M. Furthermore, the complement of C is

$$\overline{C} = \{(z, z) \mid z \in \mathbb{N}\} = C(\mathbf{0}, \{(1,1)\}),$$

and can hence be computed in polynomial time.

Now we have that $\mathcal{A}$ is universal if and only if M is a negative instance of partition. To see this, observe that every prefix $w_1 \dots w_k \in \{a, b\}^k$ of every word $\alpha \in L_\omega(\mathcal{A})$ represents a subset M' of M with $n_i \in M'$ if and only if $w_i = a$, and hence $n_i \in M \setminus M'$ if and only if $w_i = b$. The semi-linear set C of $\mathcal{A}$ states that M' and $M \setminus M'$ are not a partition of M. Hence, the claim follows. ◀

Now we show that this is also the case for deterministic strong reset PA, presenting in parallel the strategy that we will finally adapt to show that inclusion is Π_2^P-complete in the general case. We show that complements of deterministic strong reset PA recognizable ω-languages are reachability-regular PA recognizable, and given that we can complement the semi-linear set in polynomial time, we can construct such a reachability-regular PA in polynomial time. Subsequently, we show how to test intersection emptiness of a deterministic strong reset PA and a reachability-regular PA in coNP. We begin by proving the first main ingredient.

Lemma 3.2.30. *Let $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F, C)$ be a deterministic strong reset PA. Then there is a (non-deterministic) reachability-regular PA $\mathcal{A}'$ recognizing $\overline{SR_\omega(\mathcal{A})}$. Furthermore, if C can be complemented in polynomial time, then we can compute $\mathcal{A}'$ in polynomial time.*

Proof. We assume that every state of $\mathcal{A}$ is reachable from the initial state q_0 (as we can safely remove unreachable states in polynomial time). Observe that $\mathcal{A}$ rejects an infinite word α whenever one of the following two conditions is met:

1. The unique run of $\mathcal{A}$ on α visits every accepting state just finitely often.
2. The unique run of $\mathcal{A}$ on α visits an accepting state with bad counter values.

The first condition is ω-regular, while the second one can be tested with a reachability PA. We begin by clarifying the first point. Testing whether there is an infinite word rejected by $\mathcal{A}$ because its unique run visits every accepting only finitely often boils down to testing whether the complement ω-language of the underlying Büchi automaton of $\mathcal{A}$ is non-empty. Hence, let $\mathcal{B}$ be the Büchi automaton obtained from $\mathcal{A}$ by forgetting all vectors and let $\overline{\mathcal{B}}$ be a Büchi automaton recognizing the complement of $L_\omega(\mathcal{B})$. As $\mathcal{B}$ is deterministic, we can construct $\overline{\mathcal{B}}$ in polynomial time [Kur87]. Then $\overline{\mathcal{B}}$ accepts all words rejected by $\mathcal{A}$ due to the first condition.

Second, we show how to construct a (non-deterministic) reachability PA accepting all infinite words that are rejected by $\mathcal{A}$ due the second condition. Recall the following definition from Lemma 3.1.35. For any two states $p, q \in Q$ let $\mathcal{A}_{p\Rightarrow q} = (Q \cup \{q_0'\}, \Sigma, q_0', \Delta_{p\Rightarrow q}, \{q\}, C)$ where

$$\Delta_{p\Rightarrow q} = \{(q_1, a, \mathbf{v}, q_2) \mid (q_1, a, \mathbf{v}, q_2) \in \Delta, q_1 \notin F) \cup \{(q_0', a, \mathbf{v}, q_2) \mid (p, a, \mathbf{v}, q_2) \in \Delta\}.$$

For every pair of accepting states $p, q \in F$, we consider the PA $\overline{\mathcal{A}}_{p\Rightarrow q}$, which is defined as $\mathcal{A}_{p\Rightarrow q}$ but with the complement semi-linear set $\overline{C}$ of C. Observe that we can compute $\overline{\mathcal{A}}_{p\Rightarrow q}$ in polynomial time if C can be complemented in polynomial time. Hence, $\overline{\mathcal{A}}_{p\Rightarrow q}$ accepts all finite words collecting a vector not in C when starting in p, ending in q, and not visiting other accepting states in-between. Now let $\mathcal{A}^\circ_{p\Rightarrow q}$ the PA obtained from $\mathcal{A}$ and $\overline{\mathcal{A}}_{p\Rightarrow q}$ as follows. We start with a copy of $\mathcal{A}$ but every state is not accepting and every vector is replaced by $\mathbf{0}$. Then, we identify the state p in $\mathcal{A}^\circ_{p\Rightarrow q}$ with the initial state of $\overline{\mathcal{A}}_{p\Rightarrow q}$, that is, we remove every outgoing transition of p in $\mathcal{A}^\circ_{p\Rightarrow q}$ and replace them with the outgoing transitions of the initial state of $\overline{\mathcal{A}}_{p\Rightarrow q}$. Furthermore, the accepting state q of $\overline{\mathcal{A}}_{p\Rightarrow q}$ is the only accepting state of $\mathcal{A}^\circ_{p\Rightarrow q}$. Observe that q has no outgoing transitions by construction. Finally, we add a trivial self-loop to q and choose $\overline{C}$ to be the semi-linear set of $\mathcal{A}^\circ_{p\Rightarrow q}$. Hence, $\mathcal{A}^\circ_{p\Rightarrow q}$ accepts all infinite words of the form $uv\beta$ with $\beta \in \Sigma^\omega$ such that $\mathcal{A}$ is in state p after reading u, then in state q after further reading v, and collects a vector not in C while reading v and hence rejects every infinite word with prefix uv. We compute the PA $\mathcal{A}^\circ_{p\Rightarrow q}$ for every $p, q \in F$ and let $\mathcal{A}^\circ$ be the reachability PA recognizing the union of all these $\mathcal{A}^\circ_{p\Rightarrow q}$ by taking the disjoint union and connecting them with a fresh initial state (see [GJLZ22, Lemma 6] for details). In contrast to an iterated product construction, the presented construction allows us to compute $\mathcal{A}^\circ$ in polynomial time, albeit not preserving determinism.

Finally, let $\mathcal{A}'$ be a reachability-regular PA accepting $L_\omega(\overline{\mathcal{B}}) \cup R_\omega(\mathcal{A}^\circ)$. Note that $\mathcal{A}'$ can be computed in polynomial time in the sizes of $\overline{\mathcal{B}}$ and $\mathcal{A}^\circ$ by turning both automata into reachability-regular PA and again taking their disjoint union with a fresh initial state. Now we have $RR_\omega(\mathcal{A}') = \overline{SR_\omega(\mathcal{A})}$. ◀

Before we proceed, we show two auxiliary lemmas.

Lemma 3.2.31. *Let $\mathcal{A}_1 = (Q_1, \Sigma, p_I, \Delta_1, F_1, C)$ be a deterministic strong reset PA and $\mathcal{B} = (Q_2, \Sigma, q_I, \Delta_2, F_2)$ be a Büchi automaton. Then $SR_\omega(\mathcal{A}_1) \cap L_\omega(\mathcal{B})$ is ultimately periodic.*

Proof. Assume $SR_\omega(\mathcal{A}_1) \cap L_\omega(\mathcal{B}) \neq \emptyset$ and let α be an infinite word accepted by both automata, $\mathcal{A}_1$ and $\mathcal{B}$. If $\alpha = uv^\omega$ for some $u, v \in \Sigma^*$, we are done. Hence assume that this is not the case. Let $\mathcal{A} = (Q_1 \times Q_2, \Sigma, (p_I, q_I), \Delta, F_1 \times Q_2, C)$ with $\Delta = \{((p,q), a, \mathbf{v}, (p', q')) \mid (p, a, \mathbf{v}, p') \in \Delta_1, (q, a, q') \in \Delta_2\}$ be the product automaton[2] of $\mathcal{A}_1$ and $\mathcal{B}$. As $\alpha \in SR_\omega(\mathcal{A}_1)$, the unique accepting run of $\mathcal{A}_1$ on α, say $(p_0, \alpha_1, \mathbf{v}_1, p_1)\ (p_1, \alpha_2, \mathbf{v}_2, p_2) \dots$ with $p_0 = p_I$, is accepting. Likewise, as $\alpha \in L_\omega(\mathcal{B})$, there is an accepting run $(q_0, \alpha_1, q_1)(q_1, \alpha_2, q_2) \dots$ with $q_0 = q_I$ of $\mathcal{B}$ on α.

Hence, $r = ((p_0, q_0), \alpha_1, \mathbf{v}_1, (p_1, q_1))((p_1, q_1), \alpha_2, \mathbf{v}_2, (p_2, q_2)) \dots$ is a run of $\mathcal{A}$ on α with the following properties:

- there is $p_f \in F_1$ such that for infinitely many i we have $p_i = p_f$. Let $f_1, f_2, \dots$ denote the positions of all occurrences of a state of the form $(p_f, \cdot)$ in r.
- there is $q_f \in F_2$ such that for infinitely many i we have $q_i = q_f$.

Let $j \geq f_1$ be minimal such that $q_j = q_f$. Now let $k \leq j$ be maximal such that $k = f_\ell$ for some $\ell \geq 1$. Then $r[1 : f_\ell]r[f_\ell + 1 : f_{\ell+1}]^\omega$ is an accepting run of $\mathcal{A}$ on an ultimately periodic word, say uv^ω, that visits an accepting state of $\mathcal{B}$ infinitely often. Hence $uv^\omega \in SR_\omega(\mathcal{A}_1) \cap L_\omega(\mathcal{B})$. ◀

Lemma 3.2.32. *Let $\mathcal{A}_1 = (Q_1, \Sigma, p_0, \Delta_1, F_1, C)$ be a deterministic strong reset PA and let $\mathcal{B} = (Q_2, \Sigma, q_0, \Delta_2, F_2)$ be a Büchi automaton. The question $SR_\omega(\mathcal{A}_1) \cap L_\omega(\mathcal{B}) = \emptyset$ is* coNP*-complete.*

Proof. The lower bound follows immediately from the coNP-hardness of emptiness; hence, we focus on the containment in coNP by showing that testing intersection non-emptiness is in NP. By the previous lemma it is sufficient to check the existence of an ultimately periodic word. Recall the algorithm in 3.1.35 that decides the non-emptiness problem for reset PA in NP by exploiting that ω-languages accepted by a strong reset PA are ultimately periodic. We modify the algorithm as follows.

First, let $\mathcal{A}$ be the product automaton of $\mathcal{A}_1$ and $\mathcal{B}$ (as in the previous proof; however, the set of accepting states is not important for the algorithm). We guess state $p_f \in F_1$ and $q_f \in F_2$ that we expect to be seen infinitely often to satisfy the acceptance conditions of $\mathcal{A}_1$ resp. $\mathcal{B}$. Furthermore, similar to the algorithm above we guess a sequence of distinct

[2] We note that $\mathcal{A}$ interpreted as a strong reset PA does not recognize $SR_\omega(\mathcal{A}_1) \cap L_\omega(\mathcal{B})$. Instead, it accepts all infinite words $\alpha \in SR_\omega(\mathcal{A}_1)$ such that $\mathcal{B}$ has an infinite but not necessarily accepting run on α.

states $(p_1, q_1)(p_2, q_2) \dots (p_n, q_n)$ of $\mathcal{A}$ such that for some $\ell \leq n$ we have $q_\ell = q_f$, for some $k \leq \ell$ we have $p_k = p_f$, and for all $j \neq \ell$ we have $p_j \in F_1$. If $p_\ell \in F_1$, we proceed exactly as in the original NP-algorithm, that is, for every $0 \leq i < n$, we test the finite word PA $\mathcal{A}_{(p_i,q_i)\Rightarrow(p_{i+1},q_{i+1})}$, as well as the finite word PA $\mathcal{A}_{(p_n,q_n)\Rightarrow(p_k,q_k)}$ for non-emptiness.

Now assume $p_\ell \notin F_1$. Let $\mathcal{A}_{p_{\ell-1}\Rightarrow q_\ell \Rightarrow p_{\ell+1}}$ be the finite word PA that accepts all finite infixes accepted by the product automaton $\mathcal{A}$ when starting in $(p_{\ell-1}, q_{\ell-1})$, visiting (p_ℓ, q_ℓ) at some point, and ending in $(p_{\ell+1}, q_{\ell+1})$ such that for all internal states (p_i, q_i) we have $p_i \notin F_1$. Two achieve this, we take two copies of $\mathcal{A}$, where all accepting states in the first copy are not reachable, all accepting states in the second copy have no outgoing transitions, and the second copy can only be reached from the first copy via (p_ℓ, q_ℓ).

Formally, let

$$\mathcal{A}_{p_{\ell-1}\Rightarrow q_\ell \Rightarrow p_{\ell+1}} = (Q_1 \times Q_2 \times \{1,2\} \cup \{q_0'\}, \Sigma, q_0', \Delta', \{(p_{\ell+1}, q_{\ell+1})\}, C)$$

with

$$\begin{aligned}
\Delta' = {} & \{((p,q,1), a, \mathbf{v}, (p',q',1)) \mid ((p,q), a, \mathbf{v}, (p',q')) \in \Delta, p, p' \notin F_1\} \\
& \cup \{((p,q,2), a, \mathbf{v}, (p',q',2)) \mid ((p,q), a, \mathbf{v}, (p',q')) \in \Delta, p \notin F_1\} \\
& \cup \{((p,q,1), a, \mathbf{v}, (p_\ell, q_\ell, 2) \mid ((p,q), a, \mathbf{v}, (p_\ell, q_\ell)) \in \Delta\} \\
& \cup \{(q_0', a, \mathbf{v}, (p',q',1) \mid ((p_{\ell-1}, q_{\ell-1}), a, \mathbf{v}, (p',q')) \in \Delta, p' \notin F_1\} \\
& \cup \{(q_0', a, \mathbf{v}, (p_\ell, q_\ell, 2) \mid ((p_{\ell-1}, q_{\ell-1}), a, \mathbf{v}, (p_\ell, q_\ell)) \in \Delta\}.
\end{aligned}$$

Now the algorithm is similar the first case with the addition that we also test this automaton for non-emptiness. Hence, for every $0 \leq j < \ell - 1$ and $\ell + 1 \leq j < n$ we test $\mathcal{A}_{(p_j,q_j)\Rightarrow(p_{j+1},q_{j+1})}$ as well as $\mathcal{A}_{(p_n,q_n)\Rightarrow(p_k,q_k)}$ for non-emptiness. Furthermore, we test $\mathcal{A}_{p_{\ell-1}\Rightarrow q_\ell \Rightarrow p_{\ell+1}}$ for non-emptiness.

If all these tests succeed, we conclude $SR_\omega(\mathcal{A}_1) \cap L_\omega(\mathcal{B}) \neq \emptyset$, as they witness the existence of an ultimately periodic word α accepted by $\mathcal{A}_1$ with the property that there is a run an accepting run of $\mathcal{B}$ on α that visits q_f infinitely often. ◀

Let $\mathcal{A}$ be a deterministic strong reset PA whose semi-linear set can by complemented in polynomial time. By Lemma 3.2.30 we can compute a Büchi automaton $\overline{\mathcal{B}}$ and a reachability PA $\mathcal{A}^\circ$ such that $\overline{SR_\omega}(\mathcal{A}) = L_\omega(\overline{\mathcal{B}}) \cup R_\omega(\mathcal{A}^\circ)$. By the previous lemma we can test $SR_\omega(\mathcal{A}) \cap L_\omega(\overline{\mathcal{B}}) = \emptyset$ in coNP. Hence, it remains to show that testing intersection emptiness of a deterministic strong reset PA and a reachability PA is decidable in coNP. To achieve that, we use the NP-algorithm in [HZ21] deciding the reachability problem for $\mathbb{Z}$-VASS with k nested zero-tests ($\mathbb{Z}$-VASS$_k^{\text{nz}}$). A $\mathbb{Z}$-VASS$_k^{\text{nz}}$ (of dimension $d \geq 1$) is a tuple $V = (Q, Z, E)$ where Q is a finite set of states, $Z \subseteq \{0, 1, \dots, d\}$ is its set of zero tests with $|Z \setminus \{0\}| = k$, and $E \subseteq Q \times \mathbb{Z}^d \times Z \times Q$ is a finite set of transitions. A configuration of V is a pair $(p, \mathbf{v}) \in Q \times \mathbb{Z}^d$. Assume $\mathbf{v} = (v_1, \dots, v_d)$. We write $(p, \mathbf{v}) \vdash_V (p', \mathbf{v}')$ if there is a

transition $(p, \mathbf{u}, \ell, p') \in E$ such that $\mathbf{v}' = \mathbf{v} + \mathbf{u}$ and $v_1 = \cdots = v_\ell = 0$. Furthermore, we write $(p, \mathbf{v}) \vdash_V^* (p', \mathbf{v}')$ if there is a sequence $(p_1, \mathbf{v}_1) \vdash_V \cdots \vdash_V (p_n, \mathbf{v}_n)$ for some $n \geq 1$ such that $(p, \mathbf{v}) = (p_1, \mathbf{v}_1)$ and $(p', \mathbf{v}') = (p_n, \mathbf{v}_n)$. Intuitively, a $\mathbb{Z}$-VASS$_k^{\text{nz}}$ is a counter machine with zero-tests that can only zero-test the top-most ℓ counters at once, for k different values of ℓ.

The *reachability* problem for $\mathbb{Z}$-VASS$_k^{\text{nz}}$ is defined as follows: given a $\mathbb{Z}$-VASS$_k^{\text{nz}}$ V and two configurations $(p, \mathbf{v})$, $(p', \mathbf{v}')$, does $(p, \mathbf{v}) \vdash_V^* (p', \mathbf{v}')$ hold? As shown in [HZ21], this problem is NP-complete[3] for any fixed k. We are now ready to show that universality and inclusion for deterministic strong reset PA are coNP-complete if we can complement their semi-linear sets in polynomial time.

Lemma 3.2.33. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be deterministic strong reset PA with the guarantee that the semi-linear set of $\mathcal{A}_2$ can be complemented in polynomial time. Then the following questions are* coNP*-complete.*

1. *Is $SR_\omega(\mathcal{A}_2) = \Sigma^\omega$?*
2. *Is $SR_\omega(\mathcal{A}_1) \subseteq SR_\omega(\mathcal{A}_2)$?*

Proof. Hardness follows again from a reduction from partition, very similar to the reduction in Lemma 3.2.29. Hence, we focus on the containment in coNP of the second question.

As mentioned above, we can compute a Büchi automaton $\overline{\mathcal{B}}$ and a reachability PA $\mathcal{A}^\circ$ such that $\overline{SR_\omega}(\mathcal{A}_2) = L_\omega(\overline{\mathcal{B}}) \cup R_\omega(\mathcal{A}^\circ)$ by Lemma 3.2.30, and test $SR_\omega(\mathcal{A}_1) \cap L_\omega(\overline{\mathcal{B}}) = \emptyset$ in coNP by Lemma 3.2.32.

It remains to show how to solve $SR_\omega(\mathcal{A}_1) \cap R_\omega(\mathcal{A}^\circ) = \emptyset$ in coNP. In order to do so we use the NP-algorithm in [HZ21] solving reachability for $\mathbb{Z}$-VASS$_2^{\text{nz}}$ to decide $SR_\omega(\mathcal{A}) \cap R_\omega(\mathcal{A}^\circ) \neq \emptyset$ in NP. Let $\mathcal{A}_1 = (Q_1, \Sigma, p_0, \Delta_1, F_1, C_1)$ be of dimension d_1 and $\mathcal{A}^\circ = (Q_2, \Sigma, q_0, \Delta_2, F_2, C_2)$ be of dimension d_2. Assume $SR_\omega(\mathcal{A}) \cap R_\omega(\mathcal{A}^\circ) \neq \emptyset$ and let α be an infinite word accepted by both automata. In particular, there is a finite prefix u of α and an accepting run of $\mathcal{A}^\circ$ on α satisfying the Parikh condition after processing u, say in the accepting state $q_f \in F_2$. Recall that all outgoing transitions of every accepting state of $\mathcal{A}^\circ$ are self-loops, hence we may assume that $\mathcal{A}^\circ$ does not leave q_f anymore after processing u. Now let $p_f \in F_1$ be the first accepting state visited by the unique accepting run of $\mathcal{A}_1$ on α after processing u, say after processing the prefix uv. Note that $\mathcal{A}^\circ$ is still in q_f after processing uv.

Our strategy is as follows. First, we guess states p_f and q_f with the mentioned properties. We then verify these properties by building a product $\mathbb{Z}$-VASS$_2^{\text{nz}}$ with the property that $((p_0, q_0), \mathbf{0}) \vdash_V^* ((p_f, q_f), \mathbf{0})$ if and only there is a finite prefix uv as described above. Then we need to test whether $\mathcal{A}_1$ can continue a partial run from p_f to an accepting run, that is,

[3] We note that our definition of $\mathbb{Z}$-VASS$_k^{\text{nz}}$ differs slightly from the definition in [HZ21], as we allow $E \subseteq Q \times \mathbb{Z}^d \times Z \times Q$ instead of $E \subseteq Q \times \{-1, 0, 1\}^d \times Z \times Q$ only. However, this difference does not change the mentioned complexity for the reachability problem [HZ21], see also [BDG+23, Section A.1].

whether $\mathcal{A}_2$ with initial state p_f accepts at least one infinite word, say β. If all these tests succeed, the infinite word $uv\beta$ witnesses non-emptiness.

We build a product $\mathbb{Z}$-VASS$_2^{\text{nz}}$ V with $d_1 + d_2$ many counters. The idea is as follows. We use the first d_1 counters with to simulate $\mathcal{A}_1$ ensuring that every visit of an accepting state of $\mathcal{A}_1$ is with good counter values by zero-testing them. Likewise, we use a second set of d_2 counters to simulate $\mathcal{A}^\circ$. Let us give some more details on how to verify that every visit of an accepting state implies good counter values. Let $C_1 = C(\mathbf{b}_1, P_1) \cup \cdots \cup C(\mathbf{b}_k, P_k)$ for some $k \geq 1$. For every accepting state $f \in F_1$ we insert k states, say $f^{(1)}, \ldots, f^{(k)}$, and a copy of f itself. We connect f to $f^{(i)}$ with a transition subtracting $\mathbf{b}_i$ and no zero-test. Then, for every period vector $\mathbf{p}_i \in P_i$, we insert a loop on $f^{(i)}$ subtracting $\mathbf{p}_i$. Finally, every outgoing transition of $f^{(i)}$ is equipped with a zero-test on the first d_1 counters. This construction allows us to test membership of the current counter values in C_1, while resetting the counters in parallel. Finally, when reaching (p_f, q_f), we use the same idea to check whether the vector induced by the last d_2 counters yields a vector in C_2. As p_f is accepting, we expect all counters to be zero at this point, which we check by zero-testing them all. Hence, if $((p_0, q_0), \mathbf{0}) \vdash_V^* ((p_f, q_f), \mathbf{0})$ holds (where we assume that (p_f, q_f) is the state of V reached after the described zero-tests), this implies that there is a finite prefix uv such that $\mathcal{A}_2$ rejects every infinite word with prefix uv due to bad counter values, while the unique partial run of $\mathcal{A}_1$ on uv respects the strong reset acceptance condition. Finally, we need to check whether the ω-language of $\mathcal{A}_1$ with initial state p_f is not-empty using the NP-algorithm for testing emptiness for strong reset PA (see Lemma 3.1.35) If these tests succeed, there is an infinite word β such that $uv\beta \in SR_\omega(\mathcal{A}_1) \setminus SR_\omega(\mathcal{A}_2)$, witnessing non-inclusion. ◀

We are now ready to combine and generalize the results in the previous lemmas to arbitrary deterministic strong reset PA.

Lemma 3.2.34. *Universality and inclusion for deterministic strong reset PA are* Π_2^{P}*-complete.*

Proof. Hardness follows from Corollary 3.2.27; hence, we show that non-inclusion is in Σ_2^{P}, yielding the desired result.

Let $\mathcal{A}_1 = (Q_1, \Sigma, p_0, \Delta_1, F_1, C_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, q_0, \Delta_2, F_2, C_2)$ be deterministic strong reset PA of dimensions d_1 and d_2, resp. As in the previous lemma, we can compute a Büchi automaton $\overline{\mathcal{B}}$ accepting all infinite words that are rejected by $\mathcal{A}_2$ because every accepting state of $\mathcal{A}_2$ is visited only finitely often. By Lemma 3.2.32, we can test $SR_\omega(\mathcal{A}_1) \cap L_\omega(\overline{\mathcal{B}}) \neq \emptyset$ in NP. If the intersection is indeed non-empty, we conclude $SR_\omega(\mathcal{A}_1) \not\subseteq SR_\omega(\mathcal{A}_2)$. Otherwise, this non-inclusion might hold as there is an infinite word $\alpha \in SR_\omega(\mathcal{A}_1)$ rejected by $\mathcal{A}_2$ due to a reset with bad counter values. To test this case, we proceed as follows.

We guess two accepting states q_2, q_3 of $\mathcal{A}_2$ and utilize the non-irrelevance algorithm in Lemma 2.4.1 to test the existence of such a finite infix with bad counter values, i. e., an infix whose unique partial run of $\mathcal{A}_2$ yields a vector $\mathbf{v} \notin C_2$. We then construct a $\mathbb{Z}$-VASS$_2^{\text{nz}}$ to

test whether there is an infinite word accepted by $\mathcal{A}_1$ that is rejected by $\mathcal{A}_2$ because the partial run between q_2 and q_3 yields the bad vector $\mathbf{v}$, ensuring that $\mathcal{A}_1$ respects all resets, similar to the proof of the previous lemma.

Let $\mathcal{A} = (Q_1\times, Q_2, \Sigma, (p_0, q_0), \Delta, F_1 \times Q_2, C_1)$ with

$$\Delta = \{((p, q), a, \mathbf{v}, (p', q')) \mid (p, a, \mathbf{v}, p') \in \Delta_1, (q, a, \cdot, q') \in \Delta_2\}$$

be the product automaton of $\mathcal{A}_1$ and $\mathcal{A}_2$ where we only keep the vectors and accepting states from $\mathcal{A}_1$. We guess two states $(p_2, q_2), (p_3, q_3) \in Q_1 \times F_2$ such that there is $\alpha \in SR_\omega(\mathcal{A}_1) \setminus SR_\omega(\mathcal{A}_2)$ with the property that $\mathcal{A}_2$ rejects α because the unique rejecting run of $\mathcal{A}_2$ on α visits q_2 at some position, say f_2 (and hence resets), the next reset is in q_3 at position f_3, and the vector collected in this time is not a member of C_2. Furthermore, p_2 and p_3 are the corresponding states of $\mathcal{A}_1$ at positions f_2 and f_3. Additionally, we guess two states $(p_1, q_1), (p_4, q_4) \in F_1 \times Q_2$ such that the unique accepting run of $\mathcal{A}_1$ on α resets the last time before reaching position f_2 say at position $f_1 \leq f_2$, namely in p_1; and $\mathcal{A}_1$ resets the first time after reaching position f_3 say at position $f_4 \geq f_3$, namely in p_4. Furthermore, q_1 and q_4 are the states of $\mathcal{A}_2$ at positions f_1 and f_4. Observe that $f_1 = f_2$ and $f_3 = f_4$ are possible.

First, we test whether (p_1, q_1) is reachable in $\mathcal{A}$ in the sense that there is a finite prefix u of α such that $\mathcal{A}$ is in state (p_1, q_1) after reading v and $\mathcal{A}_1$ resets with good counter values whenever an accepting state of $\mathcal{A}_1$ is seen. In order to do so, we modify $\mathcal{A}$ such that (p_1, q_1) is the only accepting state, and replace its outgoing transitions with trivial self-loops. Then we use the NP-algorithm in Lemma 3.1.35 to test non-emptiness of the resulting automaton.

Second, we test whether it is possible to successfully continue the run from (p_4, q_4) in the sense that there is an infinite word β accepted by $\mathcal{A}$ when starting in (p_4, q_4) (and hence by $\mathcal{A}_1$ when starting in p_4). In order to do so, we modify $\mathcal{A}$ such that (p_4, q_4) is the initial state and test non-emptiness of the resulting automaton, again using the NP-algorithm in Lemma 3.1.35.

Let $\mathcal{A}'$ be defined as $\mathcal{A}$ but this time we only keep the vectors from $\mathcal{A}_2$ (instead of $\mathcal{A}_1$) and its semi-linear set is C_2 (instead of C_1). Third, we test whether there is indeed a finite prefix of w of α such that $\mathcal{A}'$ is in state (p_3, q_3) after processing w when starting in (p_2, q_2), and the vector collected by the unique partial run of $\mathcal{A}'$ (and hence $\mathcal{A}_2$) is not a member of C_2. To achieve that we compute the PA $\mathcal{A}'_{(p_2,q_2)\Rightarrow(p_3,q_3)}$ and test for non-irrelevance using the algorithm in Lemma 2.4.1 in Σ_2^{P} (observe that the construction of this PA preserves determinism up to completeness, and we can always complete the PA by adding a non-accepting sink). Recall that a part of this algorithm guesses a vector $\mathbf{v}$ not contained in C_2 such that there is a run ending in the accepting state (here (p_3, q_3)) collecting $\mathbf{v}$. We need to keep this vector for the next step.

Finally, we need to verify that there is a non-rejecting partial run of $\mathcal{A}$ on an infix vwx, starting in (p_1, q_1), visiting (p_2, q_2) and (p_3, q_3) in between, and ending in (p_4, q_4) such

that w is processed between the visits of (p_2, q_2) and (p_3, q_3), witnessing that $\mathcal{A}_2$ is indeed rejecting. To achieve that, we construct a product $\mathbb{Z}$-VASS$_2^{\text{nz}}$ V of dimension $d_1 + d_2$ in a similar way as in the proof of the previous lemma. We give a high level description of V. The state set of V consists of three copies of the product $Q_1 \times Q_2$, and the transitions keep the vectors of the transitions of both automata, $\mathcal{A}_1$ and $\mathcal{A}_2$. We can move from the first copy to the second copy upon reaching (p_2, q_2), and from the second copy to the third copy upon reaching (p_3, q_3). The d_2 counters belonging to $\mathcal{A}_2$ are frozen (that is $\mathbf{0}$) in the first and third copy. Contrary, all counters are used in the second copy. Furthermore, we remove every state in $Q_1 \times F_2$ in the second copy besides (p_2, q_2) and (p_3, q_3). We verify that $\mathcal{A}_1$ always resets with good counter values using zero tests as in the proof of the previous lemma. Finally, upon reaching (p_4, v_4), we subtract $\mathbf{0}^{d_1} \cdot \mathbf{v}$ (where $\mathbf{v}$ is the vector guessed in the previous step) and test whether all counters are zero. As p_4 is accepting, we expect the first d_1 counters to be zero. Furthermore, as $\mathbf{v}$ is a vector witnessing bad counter values with respect to $\mathcal{A}_2$, all-zero counters imply that there is indeed such an infix w of α breaking the run of $\mathcal{A}_2$. Let (p_f, q_f) be the state of V reached after this zero test. Then we use the NP-algorithm in [HZ21] to test $((p_1, q_1), \mathbf{0}) \vdash_V^* ((p_f, q_f), \mathbf{0})$. If the answer is positive, we conclude that $SR_\omega(\mathcal{A}_1) \not\subseteq SR_\omega(\mathcal{A}_2)$, as witnessed by $\alpha = uvwxy\beta$.

We conclude with a technical remark: the bit size of the vector $\mathbf{v}$ guessed in the third step might polynomially depend on C_1 and C_2. In particular, the bit size of (a suitable encoding of) C_1 might be arbitrary larger than the bit size of C_2. Hence, when calling the irrelevance algorithm with $\mathcal{A}'$ (where only C_2 is present) we need to take in account that the bit size of $\mathbf{v}$ might not be polynomial in C_2 but in C_1 and C_2. ◀

Finally, we study the intersection-emptiness problems, being the core of solving existential model checking. We observe that all results translate from the non-deterministic setting. Hence, for deterministic limit PA, deterministic reachability-regular PA and deterministic Büchi PA we conclude coNP-completeness by Lemma 3.1.41 and Lemma 3.1.42.

Corollary 3.2.35. *Intersection-emptiness for deterministic limit PA, deterministic reachability-regular PA, and deterministic Büchi PA is* coNP*-complete.*

For deterministic strong reset PA (and hence for deterministic weak reset PA) we obtain undecidability, as the PA constructed in the proof of Lemma 3.1.43 is indeed deterministic.

Corollary 3.2.36. *Intersection-emptiness for deterministic strong and weak reset PA is undecidable.*

4 Conclusion

We have studied the expressiveness, closure properties and classical decision problems of the non-deterministic and determinstic variants of the newly introduced models of PA operating on infinite words, namely reachability-regular PA, limit PA, weak reset PA, and strong reset PA. Notably, deterministic limit PA are closed under the Boolean operations and hence all common decision problems are decidable for them, including the classical model checking problems. Additionally, (strong) reset PA, being a very expressive model, enjoy a decidable emptiness problems as well as a decidable existential safety model checling problem. Closely related to model checking problems are synthesis problems. Here, the problem is to generate a model from a system specification (which is correct by construction). Gale-Stewart games play a key role in solving such synthesis problems [GS53]. However, these games are undecidable when winning conditions are specified by automata whose emptiness or universality problem is undecidable. Our decidability results for deterministic limit PA raise the interesting and important question whether Gale-Stewart games can be solved when their winning condition is expressed by these automata.

In future work we further plan to study the regular separability problem for these models, that is, given two ω-languages L_1, L_2 recognized by PA operating on infinite words, is there an ω-regular language L with $L_1 \subseteq L$ and $L_2 \cap L = \emptyset$. Solving this problem can be used as an alternative approach to solving existential PA model checking, as (regular) separability implies intersection emptiness. It has already been studied for some related models, e. g., PA on finite words [CCLP17] and Büchi VASS [BMZ23]. Furthermore, it remains to classify the intersections of all incomparable models and thereby to provide a fine grained "map of the universe" for Parikh recognizable ω-languages.

Part II

Solution Discovery via Reconfiguration

5 Prelude

5.1 Introduction

In many dynamic real-world applications of decision-making problems, feasible solutions must be found starting from a certain predetermined system state. This is in contrast to the classical academic problems where we are allowed to compute a feasible solution from scratch. However, when constructing a new solution from scratch, we have no control over the difference between the current system state and the new target one. Very prominent examples for such systems are typically settings where humans are involved in the system and big changes to the running system are not easily implementable or even accepted. When optimizing public transport lines, shift plans, or when assigning workers to tasks it is clearly desirable to aim for an optimal solution that is as similar as possible to the current state of the system. In this part, we develop a new framework where we aim to restore a healthy system state from an infeasible or corrupted state via a bounded number of small modification steps.

As a motivating example for our framework consider the mobile phone coverage of a city. Mobile transmission towers are placed systematically in order to ensure that every part of the city is in range of at least one such tower, connecting that part to the mobile communications network. By representing all relevant parts of the city as vertices of a graph, and connecting two vertices whenever they are close enough such that placing a mobile transmission tower in one of the parts ensures connectivity of the other part, a placement of these towers such that the whole city is connected corresponds to a dominating set. Now assume that mobile transmission towers are already installed around the city but a newly built development area in the city is not connected to the network yet. An approach in order to connect the new area could be the computation of a new dominating set from scratch. However, in general the new dominating set can differ arbitrarily from the existing one. Given the costs to move around the towers (or to disassemble them and assemble them elsewhere), it is desirable to be able to compute a new dominating set in the modified system that is close to the original one. One could also think of simply building a new tower in the new area. Albeit being a valid strategy, given the cost of building a new tower it might be desirable to re-use the already existing towers. To substantiate the example, consider the map of Berlin in Figure 5.1. The black dots on the map represent different locations[1] in Berlin, and the red squares represent mobile transmission towers together with their range. The blue triangle named v is a new development area in the district of Köpenick and, particularly, not in range of any tower. One simple solution to include v into mobile communications network is to move the tower from position u to v without changing any other tower.

[1] In fact, they are 52 important locations in Berlin according to Martin Grötschel.

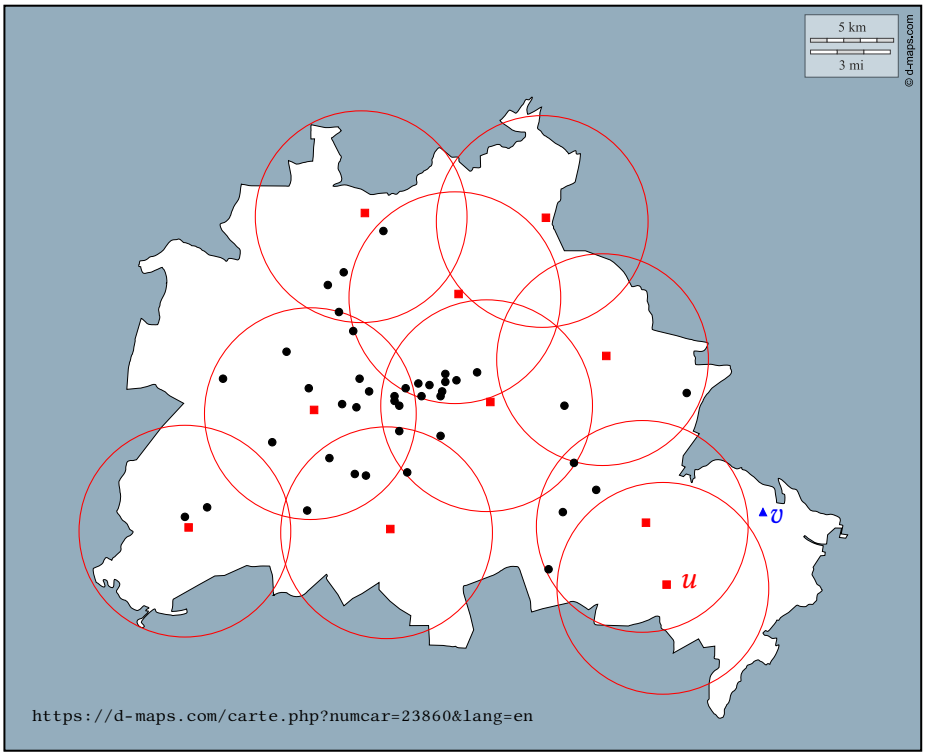


Figure 5.1. Example application of the *solution discovery via reconfiguration* framework. By moving the mobile transmission tower from u to v, we can integrate v into the mobile communication network while inducing only a small change to the current state.

To address the computational aspects of such situations, we introduce the new framework of *solution discovery via reconfiguration*. In this model, an optimizer is given a problem instance together with a current (generally infeasible) state. The goal is to decide whether a feasible solution to the given problem can be constructed by applying only a bounded number of changes to the current state. We exemplify our framework by studying the *discovery variants* of fundamental graph problems, namely Vertex Cover, Independent Set, and Dominating Set, being canonical NP-complete problems, as well as Vertex Cuts and Edge Cuts, being classical problems solvable in polynomial time. When a base problem is polynomial-time solvable, one may efficiently compute an optimal solution from scratch. However, as previously illustrated, there are situations in which a solution that is close to a currently established configuration is more desirable. As we show in this work, the constraints put on these problems in the solution discovery framework, namely a limited number of changes, can drastically alter their complexities.

Let us give more details on our framework. For any of the aforementioned problems, say L, we consider instances consisting of a graph G, a budget b, and a starting configuration of k tokens, which is not necessarily a feasible solution for L (and where tokens either occupy vertices or edges of G). The goal is to decide whether we can transform the starting configuration of tokens into a feasible solution for L by applying at most b local changes. Depending on the setting, such a local change may be the *slide* of a token to an unoccupied neighboring vertex or edge, the *jump* of a token to another unoccupied vertex or edge, or the addition or removal of a token, yielding the *token sliding* model, *token jumping* model or *token addition/removal* model, respectively. We refer to Section 5.2 for a formal definition of the framework.

Solution discovery problems are inspired by related approaches transforming one solution to another such as *local search*, *reoptimization*, *dynamic algorithms*, and *combinatorial reconfiguration*.

Local search is an algorithmic paradigm that is based on the iterative improvement of solutions by searching for a better solution in a well-defined neighborhood (typically defined by certain local changes). A local search procedure terminates once there is no improved solution in the neighborhood. Strongly related is the framework of *reoptimization* where we look for solutions for a problem instance when given solutions to neighboring instances. Local search and reoptimization have proven to be very powerful for many problems in theory and practice, see e. g., [AL97, BHK18, GH06].

In contrast to these models, in a solution discovery problem, a sequence of reconfiguration steps may not necessarily find improved solutions at each step, as long as we arrive at a feasible solution within a bounded number of steps.

In the area of *dynamic (graph) algorithms*, the dynamic nature of real-world systems is modeled via a sequence of additions and removals of vertices or edges in a graph. The goal is to output an updated solution much more efficiently than a static algorithm that is computing

one from scratch. Dynamic graph algorithms have been studied extensively, e. g., for Vertex Cover [BHI18, OR10] and Independent Set [AOSS18]. The (parameterized) complexity of dynamic algorithms has been studied, e. g., in [AKEF+15, HN13, KST18]. In contrast to the solution discovery framework, changes to the system happen in a very specific way, while the reconfiguration steps are restricted only indirectly by the computation time (and space) and not by a concrete reconfiguration budget.

In the *combinatorial reconfiguration* framework [Heu13, Nis18], we are given a graph with both, an initial solution and a target solution, and we aim to transform the initial solution to the target one by executing a sequence of reconfiguration steps chosen from a well-defined restricted collection of steps that is typically described by allowed token moves. Contrary, in a solution discovery problem, the target solution is not known (and might not even exist) and the goal is to find any feasible target solution via arbitrary intermediate configurations whose number is bounded by the budget.

We observe strong connections between the solution discovery variants of our base problems and their *weighted rainbow variants* as well as their *red-blue variants with cardinality constraints*. An instance of a weighted rainbow vertex (resp. edge) selection problem consists of a weighted vertex (resp. edge) colored graph, and the solution of such an instance may not contain two vertices (resp. edges) of the same color and additionally may not collect too much weight. We show that the weighted rainbow variants of vertex cuts and edge cuts are fixed-parameter tractable (fpt) with respect to the solution size and use these results to design fpt-algorithms for the solution discovery variants of these problems with respect to the number of tokens in the token sliding model. Similarly, solving the solution discovery variant of a problem in the token jumping model boils down to solving the red-blue variant of that same problem. An instance of a red-blue vertex (resp. edge) selection problem consists of a graph where every vertex (resp. edge) is either colored red or blue and two integer parameters k and b. The goal is to find a solution of size k that contains at most b blue vertices (resp. edges).

We study the solution discovery variants of the aforementioned problems and provide a full classification of tractability vs. intractability with respect to the classical as well as the parameterized complexity in all three token models. Moreover, we prove similar results for the rainbow variants as well as the red-blue variants of the aforementioned problems, which we believe to be of independent interest. Additionally, for some hard problems we restrict the graph classes to regain tractability. We refer to Table 5.1 for an overview of the results.

	Vertex Cover	Independent Set	Dominating Set	Vertex/Edge Cut
Discovery Sliding	NP-c., FPT[k], W[1]-hard[b]	NP-c., W[1]-hard[$k+b$]	NP-c., W[1]-hard[$k+b$]	NP-c., FPT[k], W[1]-hard[b]
Discovery Jumping	NP-c., FPT[k] W[1]-hard[b]	NP-c., W[1]-hard[$k+b$]	NP-c., W[1]-hard[$k+b$]	NP-c., FPT[k], W[1]-hard[b]
Discovery Add/Rem.	NP-c., FPT[b]	NP-c., FPT[b]	NP-c., W[1]-hard[b]	in P
Rainbow	NP-c., FPT[k]	NP-c., W[1]-hard[k]	NP-c., W[1]-hard[k]	NP-c. ($*$), FPT[k] NP-c. on planar
Red-Blue	NP-c., FPT[k] W[1]-hard[b]	NP-c., W[1]-hard[$k+b$]	NP-c., W[1]-hard[$k+b$]	NP-c., FPT[k], W[1]-hard[b]

Table 5.1. Overview of our results. For the solution discovery problems, we denote by k the number of tokens and by b the budget. For the colorful problems, we denote by k the solution size and by b the bound on the number of blue elements. The NP-completeness result for rainbow cuts ($*$) was shown in [BCL20, Theorem 5.5]; we strengthen this result by showing that the problem is already NP-hard on planar graphs.

5.2 Preliminaries

In this section we present the definitions and notations that are relevant for the second part of the thesis. We repeat the relevant notions from the preliminaries in the first part for convenience. We write $\mathbb{N}$ for the set of non-negative integers including 0.

5.2.1 Turing Machines, Decidability and Complexity Theory

Turing machines are the most common model to define the notions of computability and decidability. Informally, a Turing machine is a finite automaton equipped with an infinite tape (or a constant number of such tapes) of discrete cells that can store symbols, and a read/write head (per tape) that points on one cell at a time. The Turing machine may read the symbol of the cell the head is pointing at, write a symbol in the cell, and move the head to a neighboring cell. However, when designing algorithms it is highly inconvenient to construct Turing machines implementing these algorithms. Instead, we will always give high level descriptions. For this reason, we refrain from formally defining Turing machines and refer the interested reader to the textbooks [HMU06, Koz97, Sip13]. Note that we can always represent a decision problem as a language L. As decision problems often involve abstract objects as numbers, sets of numbers, automata, and so on, we always assume that such objects are encoded as words in an appropriate way. Hence, if a decision problem is decidable, we can implement an algorithm that solves it (in a finite amount of time).

Very often we are not just interested in the question whether a problem is decidable, but in its inherent complexity, that is, the resources needed to solve it. We denote by P the complexity class of decision problems decidable in polynomial time by a deterministic Turing machine. Similarly, we denote by NP the complexity class of decision problems that are decidable in polynomial time by a nondeterministic Turing machine. Additionally, we denote by PSPACE the complexity class of decision problems decidable with polynomial space by a deterministic Turing machine. Obviously, $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$ holds. The question whether the first inclusion is strict is the most famous question in theoretical computer science. While it is widely believed that both inclusions are strict, these questions are open for more than 50 years [Coo71].

A *polynomial time reduction* from a problem $L \subseteq \Sigma^*$ to a problem $L' \subseteq \Gamma^*$ is a total function $f : \Sigma^* \to \Gamma^*$ computable in polynomial time by a deterministic Turing machine. We write $L \leq_{\mathsf{P}} L'$ if there is a polynomial time reduction from L to L'. We call a language L *hard*[2] for a complexity class C if $L' \leq_{\mathsf{P}} L$ for all $L' \in C$. If L is C-hard and contained in C, we call L

[2]Depending on the class C, the notion of polynomial time reduction must be strengthened or can be weakened. For example, when showing NLogSpace-hardness, we require logarithmic space reductions. However, the given definition of hardness holds for all complexity classes considered in this part and we will hence ignore this technicality.

C-complete. Intuitively, the notion of C-hardness yields a lower bound for the inherent complexity of a problem, while containment in C yields an upper bound. Hence, the notion of C-completeness yields a tight bound on the inherent complexity of a problem. We will make constantly use of the following lemma.

Lemma 5.2.1. *Let C be a complexity class and $L \subseteq \Sigma^*$ and $L' \subseteq \Gamma^*$ be two languages with $L \leq_{\mathsf{P}} L'$.*

- *If L' is contained in C, then L is contained in C.*
- *If L is hard for C, then L' is hard for C.*

In order to obtain a more fine-grained analysis of the inherent complexity of a problem, it might be desirable to apply a finer analysis than classifying a problem to be in P or NP-hard. The subject of *parameterized complexity* analyzes the complexity of a problem not only depending on the input size but with respect to a *parameter*, e. g. the solution size or, speaking of solution discovery variants of graph problems, the budget.

A *parameterized problem* is a language[3] $L \subseteq \Sigma^* \times \mathbb{N}$. For an instance $(w, k) \in \Sigma^* \times \mathbb{N}$, we call k the parameter. A parameterized problem L is *fixed-parameter tractable*, fpt for short, if there exists an algorithm that on input (w, k) decides in time $f(k) \cdot |(w, k)|^c$ whether $(w, k) \in L$, for a computable function f and constant c, where $|(w, k)|$ denotes the length of (an appropriate encoding) of (w, k). We denote by FPT the complexity class of all fixed-parameter tractable parameterized problems.

Let $L \subseteq \Sigma^* \times \mathbb{N}$ and $L' \subseteq \Gamma^* \times \mathbb{N}$ be parameterized problems. An fpt-*reduction* from L to L' is a reduction f with the property that there are computable functions g, h such that $f(w, k) = (w', k')$ is computable in time $g(k) \cdot |(w, k)|^c$ for some constant c and $k' \leq h(k)$. We write $L \leq_{\mathsf{fpt}} L'$ if there is an fpt-reduction from L to L'.

The *W-hierarchy* is a collection of parameterized complexity classes

$$\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \dots.$$

Similarly to the classical complexity theory, we will make constantly use of the following lemma, originally shown in [DF92].

Lemma 5.2.2. *Let $L \subseteq \Sigma^* \times \mathbb{N}$ and $L' \subseteq \Gamma^* \times \mathbb{N}$ be parameterized problems with $L \leq_{\mathsf{fpt}} L'$.*

- *If L' is* fpt*, then L is* fpt*.*
- *If L is W[1]-hard, then L' is W[1]-hard.*

Analogously to P and NP, the question whether the inclusion $\mathsf{FPT} \subseteq \mathsf{W}[1]$ is strict is open. It is widely believed that the inclusion is strict. Hence, showing intractability in the

[3] Recall that we assume an appropriate encoding of natural numbers as words over a finite alphabet; usually, the parameter is encoded in unary.

parameterized setting is usually accomplished by establishing an fpt-reduction from a W[1]-hard problem. We refer to the textbooks [CFK+15, DF99, FG06] for extensive background on parameterized complexity.

5.2.2 Graphs

An (undirected) graph G consists of its vertex set $V(G)$ and edge set $E(G)$, where $E(G)$ is a subset of all two element sets of $V(G)$. Very often we denote an edge connecting u and v by uv instead of $\{u, v\}$. Observe that $uv = vu$ for every edge $uv \in E(G)$. If $uv \in E(G)$, we say that u and v are *adjacent*. We assume that all graphs are simple, i. e., that they do not have an edge of the form vv. The *degree* of a vertex v is the number of edges $uv \in E(G)$ for some $u \in V(G)$. A graph H is a subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A *(simple) path* (of length $k-1$) from a vertex u to a vertex v in G is a sequence of pairwise distinct vertices $v_1 \dots v_k$ such that $v_1 = u$, $v_k = v$, and $v_i v_{i+1} \in E(G)$ for all $1 \leq i < k$. Similarly, a *(simple) cycle* (of length k) in G is a sequence of pairwise distinct vertices $v_1 \dots v_k$ such that $v_i v_{i+1} \in E(G)$ for all $1 \leq i < k$ and $v_k v_1 \in E(G)$. The distance $\mathrm{dist}_G(u, v)$ between two vertices $u, v \in V(G)$ is the length of a shortest path starting in u and ending in v in G. We say a graph H is obtained from a graph G by *subdividing* an edge $e \in E(G)$ if H is obtained from G by replacing the edge e by a path of length k for some $k > 1$. A graph is *d-degenerate* if it can be reduced to the empty graph by iteratively removing a vertex of degree at most d. For example, forests (that is, acyclic graphs) are 1-degenerate. A graph is *planar* if it can be embedded into the plane without any edges crossing. Furthermore, we call a graph G *bipartite* if it does not contain cycles of odd length, or, equivalently, if its vertex set $V(G)$ can be partitioned into two parts, say V_1, V_2, such that all edges of G have one endpoint in V_1 and one endpoint in V_2. In this case, we call (V_1, V_2) a *bipartition* of G. For a vertex subset $S \subseteq V(G)$, we denote by $G[S]$ the subgraph of G *induced by* S, i. e., the graph with vertex set S and edge set $\{uv \in E(G) \mid u, v \in S\}$. Similarly, for such a set we denote by $G - S$ the subgraph of G obtained by removing the vertices of S in G, i. e., the graph $G[V(G) \setminus S]$.

Let C be a set of colors. An edge coloring is a function $\varphi : E(G) \to C$ mapping each edge $e \in E(G)$ to a color $\varphi(e) \in C$. Similarly, a vertex coloring assigns colors to vertices. An edge weight function is a function $w : E(G) \to \mathbb{N}$, and similarly a vertex weight function assigns weights to vertices. We denote weighted graphs by tuples (G, w), colored graphs by tuples (G, φ), and colored weighted graphs by tuples (G, φ, w). The weight of a set of vertices/edges is the sum of the weights of its elements.

5.2.3 Solution Discovery

Let G be a graph. A *configuration* of G is either a subset of its vertices or a subset of its edges. We formalize the notions of token moves. Under the *token addition/removal* model, a

configuration C' can be obtained (in one step) from C, written $C \vdash C'$, if $C' = C \cup \{x\}$ for an element $x \notin C$, or if $C' = C \setminus \{x\}$ for an element $x \in C$. Under the *token jumping* model, a configuration C' can be obtained (in one step) from C if $C' = (C \setminus \{y\}) \cup \{x\}$ for elements $y \in C$ and $x \notin C$. Under the *token sliding* model, a configuration C' can be obtained (in one step) from C if $C' = (C \setminus \{y\}) \cup \{x\}$ for elements $y \in C$ and $x \notin C$ if x and y are neighbors in G, that is, if $x, y \in V(G)$, then $xy \in E(G)$; and if $x, y \in E(G)$, then $x \cap y \neq \emptyset$. If C' can be obtained from C (in any model), we write $C \vdash C'$. A *discovery sequence* of length ℓ in G is a sequence of configurations $C_0C_1 \dots C_\ell$ of G such that $C_i \vdash C_{i+1}$ for all $0 \leq i < \ell$.

Let L be a vertex (resp. edge) selection problem, i. e., a problem defined on graphs such that a solution consists of a subset of vertices (resp. edges) satisfying certain requirements. The L-Discovery problem is defined as follows. We are given a graph G, a subset $S \subseteq V(G)$ (resp. $S \subseteq E(G)$), and a budget b (as a non-negative integer). The goal is to decide whether there exists a discovery sequence $C_0C_1 \dots C_\ell$ in G for some $\ell \leq b$ such that $S = C_0$ and C_ℓ is a solution for L. In general, we denote an instance of L-discovery by (G, S, b). Throughout this part, we use the variable k to denote the size of S, that is, the number of tokens.

Note that for discovery problems under the token sliding model we can always assume that $b \leq kn$, where n is the number of vertices in the input graph. This follows from the fact that each token will have to traverse a path of length at most n to reach its target position. For discovery problems in the token jumping model we can always assume $b \leq k$, as it is sufficient to move every token at most once. Similarly, for the token addition/removal model we can always assume that $b \leq n$ for vertex selection problems and $b \leq m$ for edge selection problems, where m is the number of edges in the input graph. As k is trivially upper-bounded by n for vertex selection problems (resp. m for edge selection problems), all solution discovery variants we consider are in NP and proving NP-hardness suffices to prove NP-completeness.

5.2.4 Red-Blue Variants of Vertex or Edge Selection problems

We define the *red-blue* variant of a vertex or edge selection problem in its generality, as these problems are closely related to their solution discovery variants under the token jumping model. We make some general observations, that, in particular, hold for all problems we study in this part.

Let L be an arbitrary vertex (resp. edge) selection problem. An instance of the Red-Blue-L problem consists of a colored graph (G, φ) whose vertices (resp. edges) are either colored red or blue (that is, the color set of φ is $\{\color{red}\bullet, \color{blue}\bullet\}$), as well as two non-negative integers k and b. The goal is to decide whether there exists a solution $X \subseteq V(G)$ (resp. $X \subseteq E(G)$) of L of size k such that the number of blue elements in X is at most b. We denote an instance of Red-Blue-L by (G, φ, k, b).

Lemma 5.2.3. *Let L be an arbitrary vertex (resp. edge) selection problem. There is an* fpt*-reduction from L-Discovery in the token jumping model to Red-Blue-L preserving the parameter computable in linear time.*

Proof. Let (G, S, b) be an instance of L-Discovery under the token jumping model. Let φ be a red-blue-coloring such that every vertex (resp. edge) in S is colored red and every other vertex (resp. edge) is colored blue. It is now easy to verify that $(G, \varphi, |S|, b)$ is an equivalent Red-Blue-L instance. ◀

Corollary 5.2.4. *Let L be an arbitrary vertex (resp. edge) selection problem. Then, the following results hold under the token jumping model:*

1. *If Red-Blue-L is in* P*, then L-Discovery is in* P*.*
2. *If Red-Blue-L is in* FPT *with respect to k or b, then L-Discovery is in* FPT *with respect to k or b.*
3. *If L-Discovery is* NP*-hard, then Red-Blue-L is* NP*-hard.*
4. *If L-Discovery is* W[1]*-hard with respect to k or b, then Red-Blue-L is* W[1]*-hard with respect to k or b.*

We remark that the other direction does not trivially hold, i. e., we can in general not consider an instance of Red-Blue-L as an instance of L-Discovery, as the number of red vertices/edges might exceed the bound k on the solution size.

5.2.5 A word on the token addition/removal model

In contrast to the token sliding and the token jumping model, the number of tokens in a discovery sequence under the token addition/removal model changes in general. This can lead to possibly unwanted side-effects. For NP-complete edge or vertex selection (monotone) minimization problems L (such as Vertex Cover or Dominating Set) of a graph G, the question whether there is an fpt-algorithm for L-Discovery with respect to parameter k yields the following setting. When starting with initial configuration $S = \emptyset$ (hence $k = 0$), we ask for an algorithm computing an arbitrary solution for L in time $f(0) \cdot O(|V(G)|^c) = O(|V(G)|^c)$ for some computable function f and a constant c, i. e., we ask for a polynomial time algorithm for an NP-complete problem. Hence, we do not consider the parameter k for such problems under the token addition/removal model. To avoid such situations, we could consider another approach where we restrict the definition of the addition/removal model by adding size constraints, that is, by requiring that every configuration consists of at least ℓ_1 and at most ℓ_2 tokens. However, by choosing $\ell_1 = k - 1$ and $\ell_2 = k+1$, such problems are equivalent (up to a budget factor of 2) to the token jumping model, as every addition needs to be followed by a removal and vice versa. Hence, we stick to the given definition and focus only on parameter b.

6 Solution Discovery for NP-complete Problems

6.1 Vertex Cover Discovery

A *vertex cover* in a graph G is a set of vertices $C \subseteq V(G)$ such that every edge has at least one endpoint in C, that is, for every edge $\{u, v\} \in E(G)$ we have $C \cap \{u, v\} \neq \emptyset$. In the Vertex Cover problem we are given a graph G and an integer k and the goal is to compute a vertex cover of size at most k in G.

6.1.1 Related work

The Vertex Cover problem is one of the classical NP-complete problems [GJ79] and the textbook example of a fixed-parameter tractable problem [DF99]. The problem remains NP-hard even restricted to planar graphs of maximum degree 3. It admits a 2-approximation and is hard to approximate within a factor of $(2 - \varepsilon)$, for any $\varepsilon > 0$, assuming the unique games conjecture [KR03a]. The dynamic variant of the problem was studied, e. g., in [AKEF$^+$15].

The Vertex Cover problem is also very well studied under the combinatorial reconfiguration framework [Heu13, Nis18]. In contrast to the decision variant, Vertex Cover Reconfiguration is known to be PSPACE-complete on general graphs under the token sliding model, token jumping model, and k-token addition/removal model, where the latter is defined as in our discovery setting with the additional restriction that the difference of number of tokens in every intermediate configuration and the number of initial tokens is at most k [IDH$^+$11, KMM12]. This remains true even for several restricted graph classes, see [Wro18]. On the positive side, polynomial-time algorithms are known only for very simple graph classes such as trees [DDF$^+$14]. More positive results (which vary depending on the model) are possible if we consider the parameterized complexity of the problem. We refer the reader to [MNR$^+$17] and the references therein for more details.

6.1.2 The Sliding Model

In the Vertex Cover Discovery problem, we are given a graph G, a starting configuration S, and a non-negative integer b. The goal is to decide whether we can discover a vertex cover in G (starting from S) using at most b token slides.

We prove that Vertex Cover Discovery under the token sliding model is NP-complete even restricted to planar graphs of degree four, fixed-parameter tractable with respect to parameter k and W[1]-hard with respect to parameter b.

Theorem 6.1.1. *The Vertex Cover Discovery problem under the token sliding model is NP-complete on the class of planar graphs of maximum degree four.*

Proof. We present a reduction from the Vertex Cover problem on planar graphs of maximum degree three, which is known to be NP-complete [Lic82]. Given an instance (G, k) of the Vertex Cover problem, where G is a planar graph of maximum degree three, we construct an instance of Vertex Cover Discovery as follows. We create a new graph H initially consisting of a copy of G. Then for each vertex $v \in V(H)$, we create a new path consisting of three vertices $\{x_v, y_v, z_v\}$ and we connect v to x_v. We choose $S = \{x_v, y_v \mid v \in V(G)\}$ and set the budget to $b = k$. Note that $|S| = 2|V(G)|$. This completes the construction. It follows from the construction that H is planar and of maximum degree four. We prove that (G, k) is a positive instance of the Vertex Cover problem if and only if (H, S, b) is a positive instance of the Vertex Cover Discovery problem.

First assume that G has a vertex cover C of size at most k. For every $v \in C$, we slide the token on x_v to v in H. Since C is of size at most $k = b$, we need at most b slides. To see that the resulting set is a vertex cover of H, note that every edge $\{v, x_v\}$ is still covered by either v or x_v and the edges $\{x_v, y_v\}$ and $\{y_v, z_v\}$ are still covered by y_v. Moreover, all the other edges of H are covered since C is a vertex cover of G.

For the reverse direction assume that (H, S, b) is a positive instance of the Vertex Cover Discovery problem. Since $b = k$, we know that at most k tokens can move, i. e., be placed in $H[V(G)]$. Moreover, since the resulting configuration must be a vertex cover of H and $H[V(G)]$ is an induced subgraph of H it follows that the tokens on vertices corresponding to vertices of G must form a vertex cover of G of size at most k, as needed. ◀

Before we proceed, we observe that every graph has at most 2^k vertex covers of size k. This bound follows immediately from the standard search-tree algorithm branching into the two end points of an uncovered edge, hence yielding a binary tree of depth at most k. As every leaf node corresponds to a vertex cover, and there are at most 2^k leaves in the tree, the bound follows. For more information we refer to [HNW07, CFK+15]).

Observation 6.1.2. *Every graph G has at most 2^k vertex covers of size k.*

Now we show that the Vertex Cover Discovery problem is fixed-parameter tractable with respect to parameter k.

Theorem 6.1.3. *The Vertex Cover Discovery problem under the token sliding model is fixed-parameter tractable with respect to parameter k.*

Proof. Let (G, S, b) be an instance of the Vertex Cover Discovery problem. As observed above, there are only 2^k vertex covers of size k in G. Hence, we can enumerate them and check for every vertex cover whether it is reachable from S in at most b steps by finding a minimum weight perfect matching in a bipartite graphs.

Observe that the search tree containing all vertex covers of size at most k can be constructed in time $O(2^k|E(G)|)$, as an uncovered edge can be found in time $O(|E(G)|)$ and the total

size of the search tree is bounded by 2^{k+1}. Hence, it remains to check whether a vertex cover C of size at most k in G can be reached from S using at most b token slides. To do so, we proceed as follows. For every vertex cover C, we construct an edge-weighted complete bipartite graph $(H_{S,C}, w)$, where the bipartition of $H_{S,C}$ is (S, C). For all $u \in S$ and $v \in C$, we define $w(uv) = \mathrm{dist}_G(u, v)$.

Observe that the length of a shortest discovery sequence $C_0 \dots C_\ell$ with $S = C_0$ and $C = C_\ell$ in G is equivalent to a *minimum weight perfect matching* in H, that is, the minimum weight over all sets $M \subseteq H_{S,C}(E)$ with the property that every vertex $v \in V(H_{S,C})$ appears in exactly one edge in M. It is well-known that such a matching can be found in time $O(|V(H_{S,C})|^3) \subseteq O(|V(G)|^3)$ using, e. g., Edmonds' blossom algorithm [EK72]. If the weight of a minimum weight perfect matching in H is at most b, we conclude that we are dealing with a positive instance as the vertex cover C can be discovered from S in at most b steps. Similarly, if every vertex cover of size k in G cannot be discovered within b steps, we conclude that we are dealing with a negative instance. Overall, we obtain a total running time of $O(2^k \cdot |V(G)|^3)$. ◀

Finally, we show that the Vertex Cover Discovery problem is W[1]-hard with respect to parameter b even restricted to 2-degenerate bipartite graphs by reducing from the Clique problem. A *clique* in a graph is a set of pairwise adjacent vertices. In the Clique problem we are given a graph G and non-negative integer k and the question is to decide whether there exists a clique of size at least k in G. It is well-known that the Clique problem is NP-hard [GJ79] and its parameterized variant is W[1]-hard with respect to k [DF95].

Theorem 6.1.4. *The Vertex Cover Discovery problem under the token sliding model is W[1]-hard with respect to parameter b on the class of 2-degenerate bipartite graphs.*

Proof. We present an fpt-reduction from the Clique problem. Let (G, k) be an instance of the Clique problem. We construct a graph H from G as follows (see Figure 6.1 for an illustration). Let $n = |V(G)|$ and $m = |E(G)|$.

The vertex set $V(H)$ of H is partitioned into six vertex sets $V_1, \dots, V_6$. Let $V_1 = \{r_i \mid i \leq \binom{k}{2}\}$ and $V_2 = \{z_i \mid i \leq \binom{k}{2}\}$. The set V_3 consists of $2m\binom{k}{2}$ vertices grouped into m sets V_3^e for each $e \in E(G)$. For each $e \in E(G)$, let $V_3^e = \{g_i^e, h_i^e \mid i \leq \binom{k}{2}\}$. Finally, we define $V_4 = \{y_e \mid e \in E(G)\}, V_5 = \{x_u \mid u \in V(G)\}$ and $V_6 = \{s_u \mid u \in V(G)\}$.

Now we define the edge set $E(H)$. For each $i \leq \binom{k}{2}$, we add an edge between the vertices $r_i \in V_1$ and $z_i \in V_2$. Next for each $i \leq \binom{k}{2}$ and $e \in E(G)$, we connect the vertex $g_i^e \in V_3^e$ to the vertices $z_i \in V_2$, $h_i^e \in V_3^e$, and $y_e \in V_4$. For each edge $uv \in E(G)$, we connect the vertex $y_{uv} \in V_4$ to $x_u, x_v \in V_5$. Finally, for each $u \in V(G)$, we add an edge between the vertices $x_u \in V_5$ and $s_u \in V_6$. This completes the construction of the graph H. We define the initial configuration $S = V_6 \cup V_4 \cup \{g_i^e \in V_3 \mid e \in E(G), i \leq \binom{k}{2}\}$ of size $m + n + m\binom{k}{2}$ and we set the budget to $b = 2\binom{k}{2} + k$.

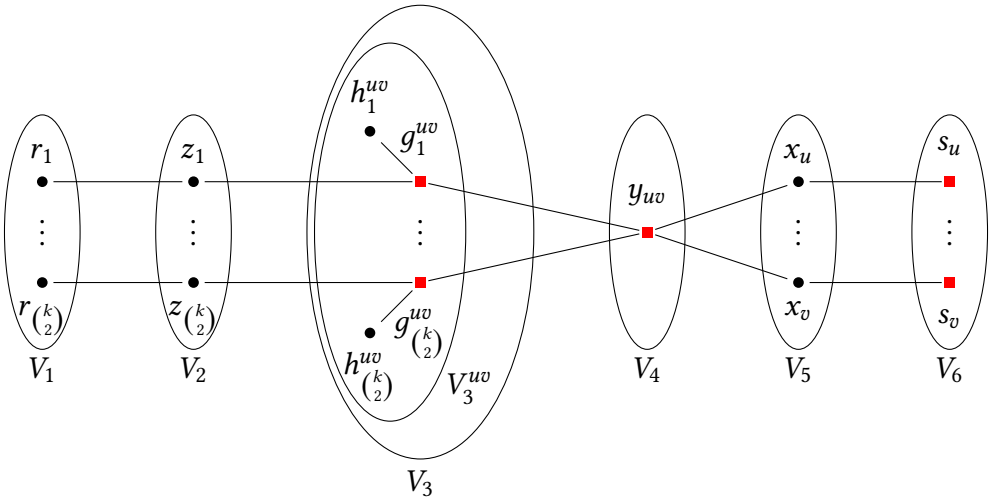


Figure 6.1. An illustration of the W[1]-hardness reduction for Vertex Cover Discovery.

We first prove that the graph H is indeed 2-degenerate. Observe that all vertices in V_1 and V_6 and the vertices h_i^e in V_3^i have a degree of one. After their removal, the vertices g_i^e in V_3^e have a degree of two. Their removal again reduces the degrees of the vertices in V_2 to one and in V_4 to 2. Finally removing the vertices in V_2 yields an edge-less graph. Note that $(V_1 \cup V_3 \cup V_5, V_2 \cup V_4 \cup V_6)$ is a bipartition of H, witnessing bipartiteness.

We claim that G has a clique of size k if and only if (H, S, b) is a positive instance of the Vertex Cover Discovery problem.

Assume that G has a clique C of size k. Consider the following vertex cover in H:

$$S' = (S \cup V_2 \cup \{x_u \mid u \in C\}) \setminus (\{y_{uv} \mid u, v \in C\} \cup \{s_u \mid u \in C\}).$$

We discover S' from S using at most b token slides as follows. For every $u, v \in C$, we slide the tokens from $g_i^{uv} \in V_3$ to $z_i \in V_2$ for arbitrary distinct values of $i \leq \binom{k}{2}$. Then we slide the tokens from $\{y_{uv} \mid u, v \in C\} \subseteq V_4$ to the vertices g_i^{uv} freed in the last step. Next we slide k tokens from $\{s_u \mid u \in C\}$ to $\{x_u \mid u \in C\}$. Since $V_2 \cup \{g_i^e \in V_3 \mid e \in E(G), i \leq \binom{k}{2}\} \subseteq S'$, the edges incident on these vertices are covered. For each $u \in V(G)$, either $x_u \in V_5$ or $s_u \in V_6$ is in S'. Hence, the edges between the sets V_5 and V_6 are covered by S'. Finally, the edges between the vertices in V_4 that are not in S' and V_5 are covered by the vertices corresponding to the clique C since the moved vertices in V_4 correspond to the edges of the clique C. Note that we moved exactly b tokens, and every token once.

Now assume that (H, S, b) is a positive instance of the *Vertex Cover Discovery* problem. As the set S does not contain any vertex from $V_1 \cup V_2$, every vertex cover in H needs to consist of $\binom{k}{2}$ vertices in $V_1 \cup V_2$ in order to cover all edges between V_1 and V_2. Observe that every discovery sequence of length at most b needs to move $\binom{k}{2}$ tokens from V_3 to V_2. As all edges

of the form $\{g_i^e, h_i^e\}$ for $e \in E(G), i \leq \binom{k}{2}$ need to be covered, $\binom{k}{2}$ vertices need to be moved from V_4 to V_3 again. After that we are left with a budget of k to cover the edges between $\binom{k}{2}$ vertices in V_4 and V_5. These edges can only be covered with k tokens (originating from V_6) if they form a clique in G. ◀

6.1.3 Jumping, Addition and Removing, and the colorful variants

Finally, we show that Rainbow Vertex Cover and Red-Blue Vertex Cover are fpt with respect to parameter k, again by exploiting Observation 6.1.2. By Corollary 5.2.4, we obtain that Vertex Cover Discovery under the token jumping model is fpt with respect to parameter k. Considering parameter b, this problem becomes W[1]-hard, as the reduction in Theorem 6.1.4 showing W[1]-hardness under the token sliding model translates directly to the jumping model. This also implies W[1]-hardness of Red-Blue Vertex Cover with respect to parameter b. Contrary to that, we show that Vertex Cover Discovery under the token addition/removal model is fpt with respect to parameter b.

First we note that all these problems remain NP-complete. The containment in NP for all these problems is obvious, while the NP-hardness for Vertex Cover Discovery under the jumping model works exactly as in Theorem 6.1.1, and the NP-hardness for the remaining problems follows from trivial reductions from Vertex Cover. Similarly, by Observation 6.1.2 (and the observation in Theorem 6.1.3), given a graph G, we can enumerate all vertex covers in G of size k in time $O(2^k|E(G)|)$. As we can test whether a vertex cover is rainbow or contains at most b blue vertices in polynomial time, we obtain containment of these problems in FPT. By Corollary 5.2.4, this result translates to Vertex Cover Discovery under the token jumping model.

Corollary 6.1.5. *The Rainbow Vertex Cover problem, the Red-Blue Vertex Cover problem, and the Vertex Cover Discovery problem under the token jumping model are* NP*-complete and fixed-parameter tractable with respect to parameter k.*

Recall the proof of Theorem 6.1.4 showing the W[1]-hardness of Vertex Cover Discovery under the token sliding model with respect to parameter b. We can re-use this reduction to show W[1]-hardness for Vertex Cover Discovery under the token jumping model. However, we only need a budget of $\binom{k}{2} + k$ (instead of $2\binom{k}{2} + k$), as we can move the vertices from the set V_4 representing the edges directly to the $\binom{k}{2}$ vertices in V_2, without the need to use to bypass the vertices in V_3. Then the core arguments translate one-to-one. This result also translates to Red-Blue Vertex Cover by Corollary 5.2.4. Hence, we obtain the following corollary.

Corollary 6.1.6. *The Red-Blue Vertex Cover problem and the Vertex Cover Discovery problem under the token jumping model are* W[1]*-hard with respect to parameter b on 2-degenerate bipartite graphs.*

Under the token addition/removal model, the Vertex Cover Discovery problem boils down to solving an instance of Vertex Cover, as clarified in the following lemma.

Lemma 6.1.7. *The Vertex Cover Discovery under the token addition/removal model is fixed-parameter tractable with respect to parameter b.*

Proof. Let (G, S, b) be an instance of the Vertex Cover Discovery problem. We may assume that every configuration sequence $C_0 \dots C_\ell$ with $C_0 = S$ yielding a vertex cover in G does never remove a token, as every superset of a vertex cover is still a vertex cover. As a consequence, we only need to compute a minimum vertex cover in the graph $G - S$ in order to obtain the smallest number of tokens needed to "complete" S to a vertex cover in G. Using an arbitrary fpt-algorithm we compute a vertex cover of size b if existent. If this is the case, we conclude that we are dealing with a positive instance, and otherwise we conclude that we are dealing with a negative instance. ◀

6.2 Independent Set Discovery

An *independent set* in a graph G is a set of vertices $I \subseteq V(C)$ that are pairwise non-adjacent, that is, for all $u, v \in I$ we have $uv \notin E(G)$. In the Independent Set problem we are given a graph G and an integer k and the goal is to compute an independent set of size at least k in G.

6.2.1 Related Work

The Independent Set problem is one of the classical NP-complete problems [GJ79], and even NP-complete to approximate within a factor of $|V(G)|^{1-\varepsilon}$, for any $\varepsilon > 0$ [Zuc06]. Despite the strong connection to the Vertex Cover problem, namely that C is a vertex cover in G if and only if $V(G) \setminus C$ is an independent set in G, the Independent Set problem is W[1]-complete with respect to parameter k [DF95].

Hence, on general graphs, local search approaches cannot be expected to improve the above stated approximation factor. However, in practice we are often dealing with graphs belonging to special graph classes, e. g., planar graphs, where local search is known to lead to much better approximation algorithms, and even to polynomial-time approximation schemes (PTAS), see e. g., [HPQ17].

The Independent Set problem is also one of the most studied problems under the combinatorial reconfiguration framework [Heu13, Nis18]. With respect to classical complexity, results for the Independent Set Reconfiguration problem and the are interchangeable (as for the base problems) and hence PSPACE-complete under the token sliding model, token jumping model, and k-token addition/removal model. More positive results (quite different than those for Vertex Cover Reconfiguration) are possible if we consider the parameterized complexity of the problem. We refer the reader to [MNR+17] and the references therein for more details.

6.2.2 The Sliding Model

In the Independent Set Discovery problem, we are given a graph G, a starting configuration S, and a non-negative integer b. The goal is to decide whether we can discover an independent set in G (starting from S) using at most b token slides.

We prove that Independent Set Discovery under the token sliding model is NP-complete even restricted to planar graphs of degree four and W[1]-hard with respect to parameter $k + b$ even restricted to graphs excluding cycles of length at least four as induced subgraphs.

Theorem 6.2.1. *The* Independent Set Discovery *problem is NP-complete on the class of planar graphs of maximum degree four.*

Proof. We present a reduction from Independent Set on planar graphs of maximum degree three, which is known to be NP-complete [Moh01]. Given an instance (G, k) with $n = |V(G)|$ of the Vertex Cover problem, where G is a planar graph of maximum degree three, we construct an instance of Independent Set Discovery as follows. We create a new graph H initially consisting of a copy of G. Then for each vertex $v \in V(H)$, we create a new path on five vertices w_v, x_v, c_v, y_v, z_v and we connect v to c_v. We choose $S = \{c_v, x_v, y_v \mid v \in V(G)\}$ and we set the budget to $b = 2n - k$, where $n = |V(G)|$. Note that $|S| = 3|V(G)|$. This completes the construction. It is easy to observe that the graph H is planar and of maximum degree four. We prove that (G, k) is a positive instance of the Independent Set problem if and only if (H, S, b) is a positive instance of the Indepedent Set Discovery problem.

First assume that G has an independent set I of size at least k. For every $v \in I$, we slide the token on c_v to v in H. For all other vertices $v \notin I$, we slide the token on x_v to w_v and the token on y_v to z_v. Observe that we need a budget of 2 to repair the path on every vertex $v \notin I$, while we need only a budget of 1 to repair the paths on vertices $v \in I$. Since I is of size at least k, we need no more than $2n - k = b$ slides. To see that the resulting set is an independent set of H, note that for every path on a vertex $v \in I$ we have moved the token from c_v to v itself. As I is an independent set, and the only conflicting neighbor of x_v resp. y_v is c_v, the tokens from these paths form an independent set. The tokens on paths of vertices $v \notin I$ also form an independent set. As the only neighbor of w_v is x_v and the token has been moved from x_v to w_v, hence there is no conflict. This is also true for y_v and z_v. As the neighbors x_v and y_v of c_v have been freed, and there is no token on v itself, the tokens on the paths form an independent set.

For the reverse direction assume that (H, S, b) is a positive instance of the Independent Set Discovery problem witnessed by the discovery sequence $C_0 \dots C_\ell$ with $S = C_0$ and $\ell \leq b$ such that C_ℓ is an independent set in H. We need to show that $|C_\ell \cap V(G)| \geq k$, which then corresponds to an independent set in G. Assume towards a contradiction that $|C_\ell \cap V(G)| = p < k$. This implies that $3n - p$ tokens are still located on the newly created paths. Since every path can contain at most 3 independent vertices and I is an independent set, at least $n - p$ paths contain 3 tokens. It takes at least 2 slides to keep the 3 tokens independent while not moving them out of the path. Hence, we require a budget of at least $2n - 2p$ for these slides. Moreover, each of the p tokens on $V(G)$ require at least one slide. In total, we require a budget of $2n - p > 2n - k$, a contradiction. ◀

Next we show that the problem is W[1]-hard with respect to parameter $k + b$ by a reduction from the Multicolored Independent Set problem, which is known to be W[1]-hard even on graphs excluding $\{C_4, \dots, C_p\}$ as induced subgraphs for any constant p [BBC+20], where C_i denotes the cycle of length i. In the Multicolored Independent Set problem we

are given a vertex-colored graph (G, φ) and an integer k, where the color set of φ consists of k colors, say $\{1, \ldots, k\}$, the question is whether there is an independent set I of size k in G such that all vertices in I have pairwise distinct colors. Note that this implies that I contains a vertex of every color.

Theorem 6.2.2. *The* Independent Set Discovery *problem is* W[1]*-hard with respect to parameter* $k + b$ *on graphs excluding* $\{C_4, \ldots, C_p\}$ *as induced subgraphs for any constant* p*.*

Proof. We present an fpt-reduction from the Multicolored Independent Set problem. Given an instance (G, φ, k) of Multicolored Independent Set with the property that G excludes $\{C_4, \ldots, C_p\}$ as induced subgraphs for any constant p, we construct an instance (H, S, b) of Independent Set Discovery as follows. First let H be a copy of G. Then, for each $i \leq k$, we add an edge on two fresh vertices $\{u_i, w_i\}$ and connect u_i to all vertices v with $\varphi(v) = i$. Observe that this construction does not add any cycles. Finally, we choose $S = \{u_i, w_i \mid i \leq k\}$ and set the budget to $b = k$. Note that $|S| = 2k$.

We claim that (G, φ, k) is a positive instance of Multicolored Independent Set if and only if (H, S, b) is a positive instance of Independent Set Discovery.

Assume that (G, φ) admits a multicolored independent set of size k. Let $I = \{v_1, \ldots, v_k\}$ denote such a set where $\varphi(v_i) = i$. Then we can discover I in H from S using b token slides by moving the token on u_i to v_i for all $i \leq k$.

For the reverse direction assume that (H, S, b) is a positive instance of the Independent Set Discovery problem witnessed by the discovery sequence $C_0 \ldots C_\ell$ with $\ell \leq b$. Observe that every edge of the form $\{u_i, w_i\}$ needs to be fixed as both endpoints contain a token. Hence, every discovery sequence needs to move the token from u_i to a vertex in $V(G)$, already exhausting the whole budget. Hence, this is only possible if (G, φ) admits a multicolored independent set. ◀

In what follows, we further investigate the parameterized complexity of the Independent Set Discovery problem with respect to b or k only instead of $k + b$ and restricted to special graph classes. First we show that the problem becomes fpt with respect to parameter k on graph classes where we can compute independence covering families in fpt-time. For a graph G and non-negative integer $k \geq 1$, a family of independent sets in G is called an *independence covering family* for (G, k), denoted by $\mathcal{F}(G, k)$, if for every independent set I in G of size at most k, there exists $J \in \mathcal{F}(G, k)$ such that $I \subseteq J$. Lokshtanov et al. [LPS+20] have shown that for many special graph classes we can compute an independence covering family for (G, k) of size at most $g(k) \cdot |V(G)|^c$ for a computable function g and a constant c using an fpt-algorithm with respect to k. These classes include, e. g., the class of all d-degenerate graphs for an arbitrary constant d.

Theorem 6.2.3. *The* Independent Set Discovery *problem is fixed-parameter tractable with respect to parameter k for every class of graphs $\mathcal{C}$ with the property that for every $G \in \mathcal{C}$ and $k \geq 1$ we can compute an independence covering family for (G, k) of size $g(k) \cdot |V(G)|^c$ for a computable function g and constant c in* fpt*-time with respect to k. In particular, the problem is fixed-parameter tractable on the class of all d-degenerate graphs for an arbitrary constant d.*

Proof. Given an instance (G, S, b) of Independent Set Discovery where $G \in \mathcal{C}$, we start by computing an independence covering family $\mathcal{F}(G, k)$ of size $g(k) \cdot |V(G)|^c$ in fpt-time, which is possible by assumption. For every $J \in \mathcal{F}(G, k)$, we construct a complete weighted bipartite graph $(H_{S,J}, w)$, where the bipartition of $H_{S,J}$ is (S, J). For all $u \in S$ and $v \in J$, we define $w(uv) = \mathrm{dist}_G(u, v)$.

Observe (similar to the graph constructed in the proof of Theorem 6.1.3) that the length of a shortest discovery sequence $C_0 \dots C_\ell$ with $S = C_0$ and $C_\ell \subseteq J$ in G is equivalent to a minimum weight matching M in $H_{S,J}$ with the property that for every $v \in S$ the set M contains an edge e with $v \in e$ (that is, M saturates S). Such a matching can be computed in time $O(|V(H_{S,J})|^3)$ using, e. g., Edmonds' blossom algorithm [EK72]. Hence, we compute the family $\mathcal{F}(G, k)$ in fpt-time and iterate over all $J \in \mathcal{F}(G, k)$ and test whether the weight of a such a minimum weight matching in $H_{S,J}$ is at most b.

We claim that (G, S, b) is a positive instance of Independent Set Discovery if and only if $(H_{S,J}, w)$ has a matching saturating S of weight at most b for at least one $J \in \mathcal{F}(G, k)$.

If (G, S, b) is a positive instance witnessed by the discovery sequence $C_0 \dots C_\ell$, then C_ℓ is an independent set in G, and in particular $C_\ell \subseteq J$ for at least one $J \in \mathcal{F}(G, k)$. Hence, the graph $H_{S,J}$ admits a matching saturating S with weight at most b, namely $\{uv \mid u \in S, v \in I\}$.

Likewise, if $H_{S,J}$ is a graph admitting a matching M saturating S with weight at most b, let $I \subseteq J$ be the set of vertices that appear as an endpoint of an edge in M. As J is an independent set in G, the set I is also an independent set in G. By the construction of $H_{S,J}$, the matching M witnesses that I can be discovered from S in at most b steps. Hence, we conclude that (G, S, b) is a positive instance of Independent Set Discovery. ◀

In contrast to the previous theorem, we show that Independent Set Discovery is W[1]-hard with respect to parameter b even on very restricted graph classes by a reduction from the Multicolored Clique problem which is known to be W[1]-hard [CFK+15]. Similar to the Multicolored Independent Set problem, in the Multicolored Clique Set problem we are given a vertex-colored graph (G, φ) and an integer k, where the color set of φ consists of k colors, say $\{1, \dots, k\}$, the question is whether there is clique C of size k in G such that all vertices in C have pairwise distinct colors.

Theorem 6.2.4. *The* Independent Set Discovery *problem is* W[1]*-hard with respect to parameter b on the class of 2-degenerate bipartite graphs.*

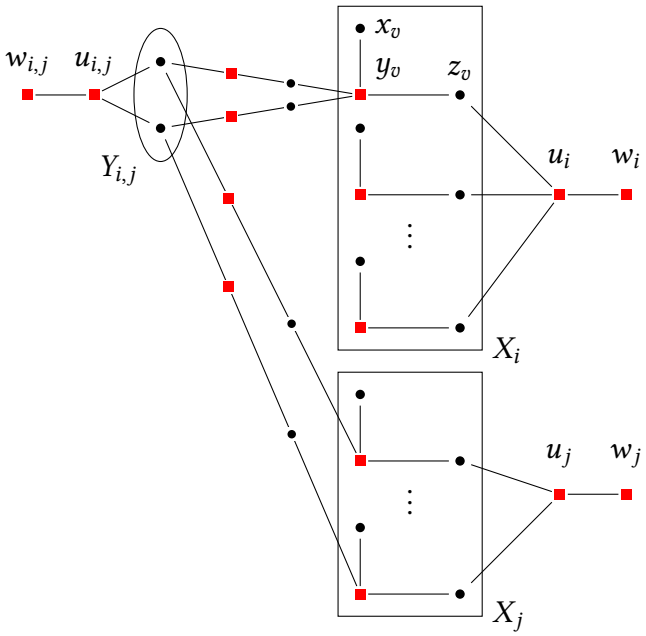


Figure 6.2. An illustration of the W[1]-hardness reduction for Independent Set Discovery on 2-degenerate bipartite graphs.

Proof. We present an fpt-reduction from the Multicolored Clique problem. Given an instance (G, φ, k) with $n = |V(G)|$ and $m = |E(G)|$ of Multicolored Clique, we construct an instance (H, S, b) of Independent Set Discovery as follows (see Figure 6.2 for an illustration).

First let H be a copy of G. We replace every $v \in V(G)$ by a path on 3 vertices which we denote by x_v, y_v, and z_v. For all $i \leq k$, we define the set $X_i = \{x_v, y_v, z_v \mid \varphi(v) = i\}$. Moreover, for all $i \leq k$, we add an edge on two fresh vertices $\{u_i, w_i\}$ and connect u_i to all vertices z_v with $\varphi(v) = i$. For $i < j \leq k$, we denote by $E_{i,j} = \{uv \in E(G) \mid \varphi(u) = i, \varphi(v) = j\}$ the set of edges connecting vertices of color i and j. For each $E_{i,j}$, we create a new set of vertices, which we denote by $Y_{i,j}$, containing one vertex e_{uv} for each edge $\{u, v\} \in E_{i,j}$. We additionally add an edge on two fresh vertices $\{w_{i,j}, u_{i,j}\}$ and connect $u_{i,j}$ to all vertices in $Y_{i,j}$. For each vertex $e_{uv} \in Y_{i,j}$, we connect e_{uv} to y_u via a path consisting of two new vertices and we connect e_{uv} to y_v via a path consisting of two new vertices.

Let $X = \bigcup_{i \leq k} X_i$ and $Y = \bigcup_{i<j\leq k} Y_{i,j}$. We denote by W the set of all vertices along paths from Y to X that are at distance one from some vertex in Y and we use Z to denote the set of all vertices along such paths that are at distance two from some vertex in Y. Finally, let

$$S = W \cup \{u_i, w_i \mid i \leq k\} \cup \{u_{i,j}, w_{i,j} \mid i < j \leq k\} \cup \{y_v \mid v \in V(G)\}$$

and set the budget to $b = 3\binom{k}{2} + 2k$. Note that $|S| = 2k + 2\binom{k}{2} + n + 2m$.

It is not hard to see that the graph H is indeed bipartite by construction. To see that the graph H is 2-degenerate it suffices to note that all vertices in W are of degree 2, and their removal yields a forest (which is 1-degenerate).

We claim that (G, φ, k) is a positive instance of Multicolored Clique if and only if (H, S, b) is a positive instance of Independent Set Discovery.

Assume that (G, φ) admits a multicolored clique of size k. Let $C = \{v_1, \dots v_k\}$ denote such a clique where $\varphi(v_i) = i$. Consider the discovery sequence where we slide the token on y_{v_i} to x_{v_i} and then slide the token on u_i to z_{v_i} for all $i \leq k$ (using a budget of $2k$). Next, for all $i < j \leq k$ we slide the token on $u_{i,j}$ to the vertex $e_{v_i v_j} \in Y_{i,j}$ and then slide the two tokens in W to their neighbors in Z (using a budget of $3\binom{k}{2}$). Since C is a multicolored clique in G, the vertices in Z with a token on it only have neighbors of the form y_{v_i} in the sets X_j. As we freed exactly these vertices in the first steps, the resulting configuration is indeed an independent set of H.

For the reverse direction, assume that (H, S, b) is a positive instance of Independent Set Discovery witnessed by the discovery sequence $C_0 \dots C_\ell$. Since we have two adjacent tokens on u_i and w_i for all $i \leq k$, all tokens on the u_i are moved (to a vertex of the form z_v) in the discovery sequence. Moreover, since every vertex y_v contains a token (and is adjacent to z_v), the vertices on the y_v need also to be moved. Hence, we need a minimum of $2k$ slides for the edges of the form $\{u_i, w_i\}$. Similarly, for each pair i, j with $i < j \leq k$, we have two adjacent tokens on $w_{i,j}$ and $u_{i,j}$. Moreover, all vertices in W contain tokens. Hence, we need at least three slides for a total of $3\binom{k}{2}$ slides for the edges of the form $\{u_{i,j}, w_{i,j}\}$. Hence, there must exist k vertices y_{v_i} and $\binom{k}{2}$ vertices $e_{v_i v_j}$ adjacent to those vertices in order to successfully slide the tokens on the u_i and $u_{i,j}$ vertices away from their neighbors w_i and $w_{i,j}$ that contain tokens. This is only possible if these vertices v_i form a clique in G which is multicolored by construction. ◀

6.2.3 Jumping, Addition and Removing, and the colorful variants

Finally, we show that the hardness results for Independent Set Discovery under the token sliding model translate to the token jumping model. In fact, we only need small modifications to re-use the constructions from the token sliding model. By Corollary 5.2.4, the hardness results translate to Red-Blue Independent Set as well.

Indeed, the hardness-reductions in Theorem 6.2.1, Theorem 6.2.2 and Theorem 6.2.4 for Independent Set Discovery under the token sliding model translate directly to the token jumping model, as the ability to jump does not alter the key observations in these proofs. In particular, in all of these proofs, we move b tokens exactly once. It is easily verified that jumping leads to the same solution discoveries. Hence, we obtain the following corollary.

Corollary 6.2.5. *The* Independent Set Discovery *problem under the token jumping model and the* Red-Blue Independent Set Discovery *problem are* NP*-complete on the class of planar graphs of maximum degree four,* W[1]*-hard with respect to parameter* $k + b$ *on graphs excluding* $\{C_4, \ldots, C_p\}$ *as induced subgraphs for any constant* p*, and* W[1]*-hard with respect to parameter* b *on the class of* 2*-degenerate bipartite graphs.*

We note that as Rainbow Independent Set is a generalization of Independent Set, all hardness results translate trivially.

Corollary 6.2.6. *The* Rainbow Independent Set *problem is* NP*-hard and* W[1]*-hard with respect to parameter* k*.*

We conclude by showing that Independent Set Discovery under the token addition/removal model boils down to solving an instance of Vertex Cover.

Lemma 6.2.7. *The* Independent Set Discovery *problem under the token addition/removal model is fixed-parameter tractable with respect to parameter* b*.*

Proof. Let (G, S, b) be an instance of the Independent Set Discovery problem. We may assume that every configuration sequence $C_0 \ldots C_\ell$ with $C_0 = S$ yielding an independent set in G does never add a token, as every subset of an independent set is still an independent set. As a consequence, it is sufficient to consider the graph $G[S]$. As every edge of $G[S]$ yields a conflict, the problem boils down to the computation of a vertex cover of size at most b in $G[S]$. Hence, we use an arbitrary fpt-algorithm to compute of vertex cover C of size b if existent. If this is the case, we conclude that we are dealing with a positive instance, as $S \setminus C$ is an independent set in G. Otherwise we conclude that we are dealing with a negative instance. ◀

6.3 Dominating Set Discovery

A *dominating set* in a graph G is a set of vertices $D \subseteq V(G)$ such that every vertex $v \in V(G)$ is contained in D or adjacent to a vertex in D, that is, we have $v \in D$ or there is a vertex $u \in D$ with $uv \in E(G)$. In the Dominating Set problem we are given a graph G and an integer k and the goal is to compute a dominating set of size at most k in G.

6.3.1 Related Work

The Dominating Set problem is one of the classical NP-complete problems [GJ79]. From the parameterized perspective, the problem is W[2]-complete with respect to parameter k and hence believed to be harder than Independent Set. However, the Dominating Set problem can be approximated up to a factor of $\log |V(G)|$ by a simple greedy algorithm [Joh73, Lov75] and this factor asymptotically cannot be improved unless P = NP [DS14].

Similar to the Independent Set problem, local search approaches cannot be expected to improve the above stated approximation factor on general graphs. However, restricting to special graph classes as the class of planar graphs, local search is known to lead to much better approximation algorithms and even PTAS [HPQ17]. Furthermore, the Dominating Set problem is known to be fixed-parameter tractable with respect to k on many special graph classes; we refer to [TV19] for an overview. The dynamic variant of the problem was studied e. g., in [AKEF$^+$15].

For the Dominating Set Reconfiguration problem, it is known that the problem is PSPACE-complete under the token sliding model, token jumping model, and k-token addition/removal model even restricted to many special graph classes, e. g., bipartite graphs or planar graphs [HIM$^+$16]. On the positive side, polynomial-time algorithms are known only for very simple graph classes such as trees.

6.3.2 The Sliding Model

In the Dominating Set Discovery problem, we are given a graph G, a starting configuration S, and a non-negative integer b. The goal is to decide whether we can discover a dominating set in G (starting from S) using at most b token slides.

We prove that Dominating Set Discovery under the token sliding model is NP-complete even restricted to planar graphs of degree five and W[2]-hard with respect to parameter $k + b$ even restricted to bipartite graphs. Furthermore, when restricting to certain graph classes, the problem is fpt with respect to parameter k but W[1]-hard with respect to parameter b on 2-degenerate graphs.

Theorem 6.3.1. *The* Dominating Set Discovery *problem is* NP*-complete on the class of planar graphs of maximum degree five.*

Proof. We present a reduction from Dominating Set on planar graphs of maximum degree three, which is known to be NP-complete [GJ79]. Given an instance (G, k) of Dominating Set, where G is a planar graph of maximum degree three, we construct an instance of Dominating Set Discovery as follows. We create a new graph H initially consisting of a copy of G. Then, for each vertex $v \in V(H)$, we create a new path on four vertices w_v, x_v, y_v, z_v and connect v to w_v. Then, we create an additional new vertex u_v and connect u_v to both v and x_v. We choose $S = \{x_v, y_v, \mid v \in V(G)\}$ and we set the budget to $b = 2k$. Note that $|S| = 2|V(G)|$. This completes the construction. It is easy to observe that the graph H is planar and of maximum degree five. We prove that (G, k) is a positive instance of the Dominating Set problem if and only if (H, S, b) is a positive instance of the Dominating Set Discovery problem.

First assume that G has a dominating set D of size at most k. For every $v \in D$ we slide the token on x_v to v in H using a budget of $2k$. To see that the resulting set is a dominating set in H, note that every pair of vertices w_v and u_v is dominated by either x_v or v. Moreover, every pair of vertices y_v and z_v is dominated by y_v. The vertex x_v is either dominated by x_v or y_v (depending on whether $v \in D$ or not). Since D is a dominating set of G, all vertices in $V(G)$ are also dominated, hence the resulting configuration yields a dominating set.

For the reverse direction assume that (H, S, b) is a positive instance of the Dominating Set Discovery problem. Note that moving a token on x_v to either w_v or u_v leaves a none dominated vertex. Moreover, to dominate all vertices $\{u_v, w_v, x_v, y_v, z_v\}$ we need at least two tokens which implies that in the resulting configuration, for each v, we either have a token on v and a token on y_v, or the from the initial configuration do not move, that is, one token on x_v and one token on y_v. Since $b = 2k$ and the distance from x_v to v is two, we can have at most k tokens slide to vertices corresponding to vertices of G. Such vertices must form a dominating set in G, as needed. ◀

Next we show that the problem is W[2]-hard with respect to parameter $k + b$ again by a reduction from Dominating Set.

Theorem 6.3.2. *The* Dominating Set Discovery *problem is* W[2]*-hard with respect to* $k + b$ *on the class of bipartite graphs.*

Proof. We present a parameterized reduction from the Dominating Set problem, which is known to be W[2]-hard on general graphs. Given an instance (G, k) of Dominating Set, we construct an instance (H, S, b) of Dominating Set Discovery as follows (see Figure 6.3 for an illustration).

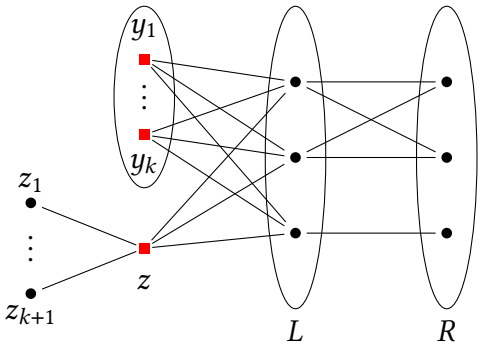


Figure 6.3. Illustration of the hardness reduction for Dominating Set Discovery.

First, H contains two copies of the vertex set of G. We denote these two sets by L and R. We connect each vertex $v \in L$ to the copy of v in R, as well as to the copies of all adjacent vertices in R, i. e., to all vertices in $\{v\} \cup \{u \mid uv \in E(G)\} \subseteq R$, that is, we connect v to each vertex in its closed neighborhood in R. Then we add k fresh vertices $y_1, \ldots, y_k$ and connect them to all vertices in L. We further add one fresh vertex z with $k+1$ pendent neighbors $\{z_1, \ldots, z_{k+1}\}$ attached to it. We connect z to all vertices in L. We choose $S = \{z, y_1, \ldots, y_k\}$ and we set the budget to $b = k$. Note that $|S| = k+1$. This completes the construction. Observe that H is indeed bipartite as witnessed by the bipartition $(L \cup \{z_1, \ldots z_{k+1}\}, R \cup \{y_1, \ldots, y_k\} \cup \{z\})$.

We claim that (G, k) is a positive instance of Dominating Set if and only if (H, S, b) is a positive instance of Dominating Set Discovery.

Assume that G has a dominating set D of size k. For all $i \leq k$, we slide the token on y_i to an arbitrary copy of a vertex of D in L. Those vertices dominate $R \cup \{y_1, \ldots, y_k\}$ while vertex z dominates $L \cup \{z_1, \ldots z_{k+1}\}$. Hence, the resulting configuration yields a dominating set in H.

For the reverse direction, assume that (H, S, b) is a positive instance of Dominating Set Discovery witnessed by the discovery sequence $C_0 \ldots C_\ell$. We can assume without loss of generality that the resulting dominating set C_ℓ contains z and no vertex of R. This follows from the fact that not having a token on z requires moving $k+1 > b$ tokens to its neighbors $\{z_1, \ldots, z_{k+1}\}$, which is not feasible. Moreover, every token in R can only dominate itself since L is already dominated by z. Hence, every token in R can instead be placed using less moves on the copy of the same vertex in L and potentially dominate more vertices. Putting it all together, we can assume that C_ℓ contains a token on z and at most k tokens in L that must dominate all vertices of R. This implies that $C_\ell \setminus \{z\}$ is a dominating set of size k in G, as needed. ◀

For the next result, we use the standard reduction from Vertex Cover to Dominating Set to reduce Vertex Cover Discovery on 2-degenerate bipartite graphs (which is W[1]-hard with respect to b by Theorem 6.1.4) to Dominating Set Discovery on 2-degenerate graphs.

Theorem 6.3.3. *The* Dominating Set Discovery *problem is* W*[1]-hard with respect to parameter b on the class of 2-degenerate graphs.*

Proof. Given an instance (G, S, b) of Vertex Cover Discovery, we create an instance (H, S, b) of Dominating Set Discovery where H is obtained from G by adding a new vertex e_{uv} for each edge $uv \in E(G)$ and connecting e_{uv} to both u and v. Observe that we do not increase the distances between vertices of G in H and that we can assume, without loss of generality, that a dominating set of H does not contain one of the newly introduced vertices of the form e_{uv} (as picking either u or v dominates at least the vertices dominated by e_{uv}). Note that since we start with a 2-degenerate bipartite graph G and we simply add vertices of degree two (forming triangles) it follows that H is also 2-degenerate (but not bipartite). This concludes the proof. ◀

Finally, we show that the problem becomes fpt with respect to parameter k on graph classes where we can compute domination cores in fpt-time. For a graph G and non-negative integer $k \geq 1$, a set $C \subseteq V(G)$ is a *k-domination core* for G if every set of size at most k that dominates C also dominates G. Hence, given a k-domination core for G, we can compute a dominating set of size k by computing a set that only needs to dominate C. Given that C is small, this leads to fpt-algorithms for Dominating Set. Although it is very unlikely that domination cores of small sizes exist for general graphs, it is known that they exist for many special graph classes, e. g. planar graphs. We refer to [DK09, KRS18, TV19] for more information. We show that Dominating Set Discovery is fixed-parameter tractable with respect to parameter k on every class of graphs where we can compute k-domination cores in fpt-time (with respect to k).

Theorem 6.3.4. *The* Dominating Set Discovery *problem is fixed-parameter tractable with respect to parameter k for every class $\mathcal{C}$ of graphs with the property that for every $G \in \mathcal{C}$ and $k \geq 1$ we can compute a k-domination core of size $g(k) \cdot |V(G)|^c$ for a computable function g and constant c in* fpt*-time with respect to k. In particular, the problem is fixed-parameter tractable on planar graphs.*

Proof. Given an instance (G, S, b) of Dominating Set Discovery where $G \in \mathcal{C}$, we start by computing a k-domination core C for G of size $g(k) \cdot |V(G)|^c$ in fpt-time, which is possible by assumption. We then compute the projection classes towards C, that is, we compute a family of sets of vertices such that any two vertices $u, v \in V(G) \setminus C$ belong to the same set if and only if $N(u) \cap C = N(v) \cap C$, where $N(u) = \{v \mid uv \in E(G)\}$ denotes the open neighborhood of vertex u. The number of projection classes is trivially upper bounded by $2^{g(k)}$, hence dependent on k only. By the definition of domination cores, we can assume, without loss of generality, that every minimal dominating set of G of size at most k contains at most one vertex from each projection class. Hence, we can now enumerate all minimal dominating sets of size at most k by treating each projection class as a single

vertex. Since both $|C|$ and the number of projection classes is bounded by a function of k, this brute-force enumeration can be accomplished in time bounded by some function of k. Let D denote a dominating set consisting of vertices from C as well as vertices representing projection classes. We now construct a complete weighted bipartite graph $(H_{S,D}, w)$, where the bipartition of H is (S, D). For all $u \in S$ and $v \in D \cap C$, we define $w(uv) = \text{dist}_G(u, v)$. Furthermore, for all $u \in S$ and $v \in D \setminus C$, we define $w(uv)$ to be the shortest distance from u to any vertex in the corresponding projection class. In particular, we have $w(uv) = 0$ if u belongs to the same class as v. The rest of the proof is identical to the proof of Theorem 6.2.3 (see also Theorem 6.1.3), that is, we look for a matching saturating S of weight at most b in $H_{S,D}$. If such a matching exists for at least one D, we conclude that we are dealing with a positive instance. Otherwise we conclude that we are dealing with a negative instance. ◀

6.3.3 Jumping, Addition and Removing, and the colorful variants

Similar to the Independent Set Discovery problem, we show that the hardness results for Dominating Set Discovery under the token sliding model translate to the token jumping model. In fact, we only need small modifications to re-use the constructions from the token sliding model. By Corollary 5.2.4, the hardness results translate to Red-Blue Independent Set as well.

Indeed, the W[2]-hardness-reduction in Theorem 6.3.2 translates one-to-one to the jumping model by the observations in the proof. Similar, for the NP-hardness reduction in Theorem 6.3.1 we only need to change the budget to $b = k$ (instead of $b = 2k$), as the tokens on x_v can jump directly to v for a cost of one per token. Finally, for the W[1]-hardness reduction in Theorem 6.3.3, we reduce from Vertex Cover Discovery under the token jumping model, which is also W[1]-hard with respect to parameter b on 2-degenerate bipartite graphs by Corollary 6.1.6.

Corollary 6.3.5. *The Dominating Set Discovery problem under the token jumping model and the Red-Blue Dominating Set Discovery problem are* NP*-complete on the class of planar graphs of maximum degree five,* W[2]*-hard with respect to parameter $k + b$ on bipartite graphs, and* W[1]*-hard with respect to parameter b on the class of 2-degenerate graphs.*

We note that as Rainbow Dominating Set is a generalization of Dominating Set, all hardness results translate trivially.

Corollary 6.3.6. *The Rainbow Dominating Set problem is* NP*-hard and* W[2]*-hard with respect to parameter k.*

This is also the case for Dominating Set Discovery under the token addition/removal model, as an instance of such a problem with initial configuration $S = \emptyset$ is equivalent to finding a dominating set of size b. Hence, the hardness results translate to this problem as well.

Corollary 6.3.7. *The Red-Blue Dominating Set problem is* NP*-hard and* W[2]*-hard with respect to parameter* b.

7 Solution Discovery for Problems in P

7.1 Vertex Cut Discovery and Edge Cut Discovery

A *vertex cut* in a graph G between two of its vertices s and t is a set of vertices $C \subseteq V(G)$ such that every path from s to t in G contains a vertex of C. Likewise, an *edge cut* in a graph G between two of its vertices s and t is a set of edges $C \subseteq E(G)$ such that every path from s to t in G contains an edge of C. In the Vertex Cut (resp. Edge Cut) problem we are given a graph G, vertices s and t, and an integer k and the goal is to compute a vertex cut (resp. edge cut) of size at most k separating s and t in G.

7.1.1 Related Work

The Vertex Cut as well as the Edge Cut problem are both known to be polynomial time solvable, e. g., by reducing them to finding maximum flows in networks using Menger's theorem [Men27]. Then we can find such flows by utilizing the flow algorithm by Edmond and Karp [EK72].

Gomes et al. [GNS20] studied the Vertex Cut problem under the combinatorial reconfiguration framework [Heu13, Nis18] and show PSPACE-hardness under the token jumping model and NP-hardness under the token jumping model and k-token addition/removal model. These hardness results even hold on the class of bipartite graphs.

The Rainbow Edge Cut problem has already been studied together with several other rainbow disconnection problems in [BCL20]. The authors show that Rainbow Edge Cut is NP-complete on general graphs. We show that this problem remains NP-complete even if we restrict it to the class of planar graphs.

7.1.2 Rainbow Vertex Cut and Rainbow Edge Cut

This time we start with the rainbow variants in order to establish an fpt-algorithm for the discovery variants with respect to parameter k. Given an edge-colored graph (G, φ) with vertices $s, t \in V(G)$ and an integer k, the Rainbow Edge Cut problem asks whether there exists an edge cut $C \subseteq E(G)$ of size k separating s and t such that all edges in C have pairwise different colors. The Rainbow Vertex Cut problem can be defined analogously where instead of $C \subseteq E(G)$ we look for a rainbow subset of vertices $C \subseteq V(G)$ separating s and t. The Rainbow Edge Cut problem is known to be NP-complete [BCL20, Theorem 5.5]. We start by showing that the problem remains NP-complete on planar graphs by a reduction from the Rainbow Matching problem.

A *rainbow matching* in an edge-colored graph (G, φ) is a set of edges $M \subseteq E(G)$ such that each vertex $v \in V(G)$ appears in at most one edge in M and all edges in M have pairwise distinct colors. In the Rainbow Matching problem we are given an edge-colored graph

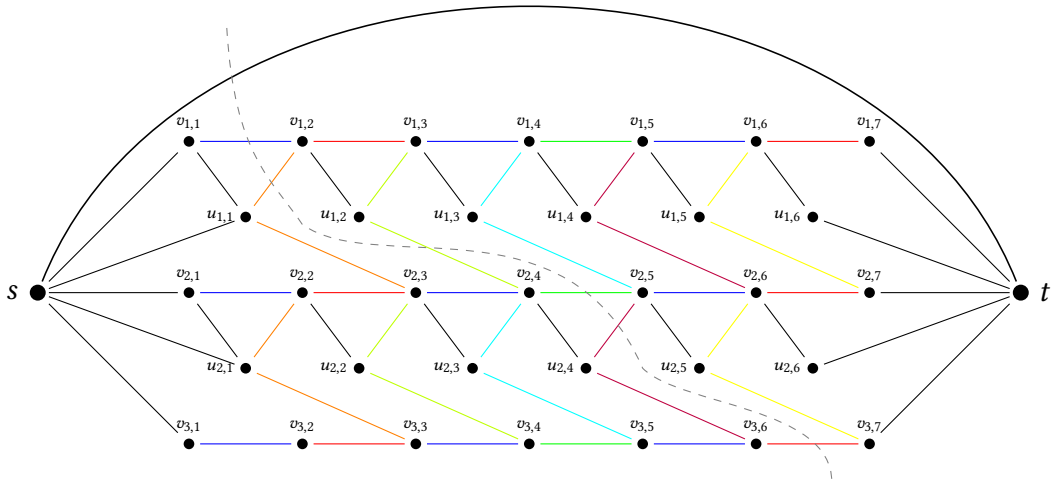


Figure 7.1. Illustration of the hardness reduction for Rainbow Edge Cut on planar graphs.

(G, φ) and an integer k and the goal is to compute a rainbow matching of size k in G. The Rainbow Matching problem is known to be NP-complete, even restricted to paths [LP14].

Theorem 7.1.1. *The Rainbow Edge Cut problem is* NP*-complete on planar graphs.*

Proof. Containment in NP is clear as Rainbow Edge Cut on general graphs is in NP. Hence we focus on the hardness proof. We present a reduction from Rainbow Matching on paths. Let (P, k) be an instance of Rainbow Matching where P is an edge-colored path on n vertices denoted by $v_1, \dots v_n$ and the edges are colored with colors from a color set C.

We construct an instance $(G, \varphi : E(G) \to C')$ of Rainbow s-t-Cut as follows. The new color set is $C' = C \cup \{\text{black}\} \cup \{c_i \mid i \leq n-2\}$, that is, C' uses the colors from C as well as $n-2$ fresh colors and the color black.

Let us describe the construction of G in detail; see Figure 7.1 for an illustration. In the first step, G consists of k disjoint copies of P, which we call $P_1, \dots, P_k$. Let the vertices of P_j be called $v_{j,1} \dots v_{j,n}$. Additionally, we add two fresh vertices s and t. For every $j < k$, insert a set L_j of $n-1$ fresh vertices, and call them $u_{j,1}, \dots, u_{j,n-1}$. Hence,

$$V(G) = \bigcup_{j \leq k} V(P_j) \cup \bigcup_{j<k} V(L_j) \cup \{s, t\}.$$

Now we connect s and t with a black edge, which enforces that this black edge must be part of every (rainbow) cut separating s and t. Hence, any other black edge may not be part of such a rainbow cut. We connect s and t to the vertices in P_j and L_j as follows. For every

$j \leq k$ we insert the edges $\{s, v_{j,1}\}$ and $\{v_{j,n}, t\}$, which are colored black. Likewise, for every $j < k$ we insert the edges $\{s, u_{j,1}\}$ and $\{u_{j,n-1}, t\}$, which are also colored black. Finally, for every $j < k$ and $i \leq n-2$, we insert the edge $\{v_{j,i}, u_{j,i}\}$ which is colored black and the edges $\{u_{j,i}, v_{j,i+1}\}$ and $\{u_{j,i}, v_{j+1,i+2}\}$, which are colored c_i. This finishes the construction.

We claim that P has a rainbow matching of size k if and only if G admits a rainbow cut separating s and t of size n. Assume that P has a rainbow matching $M = \{e_1, \ldots, e_k\}$ of size k. Without loss of generality, we assume that the edges in M are ordered with respect to the v_i, i.e., if $e_i = \{v_{\ell_i}, v_{\ell_{i+1}}\}$ and $i < j$, then $\ell_i < \ell_j$. We claim that the set C that consists of the black edge $\{s, t\}$, the copy of e_i in P_i, and the obvious edges connecting the P_i and L_j is a rainbow s-t-cut of G. To be precise, we have

$$C = \{s, t\} \cup \{\{v_{i,\ell_i}, v_{i,\ell_i+1}\}, \{u_{i,\ell_i}, v_{i,\ell_i+1}\} \mid i \leq k\} \cup \bigcup_{i \leq k} \{\{u_{i,j}, v_{i,j+2}\} \mid \ell_i < j \leq \ell_{i+1} - 2\}.$$

Observe that C is a cut by construction (see Figure 7.1), and that no two edges in C have the same color, as M is a rainbow matching, and for every $j < k$ and $i \leq n-2$ at most one of the edges $\{v_{j,i}, u_{j_i}\}$ and $\{u_{j_i}, v_{j+1,i+2}\}$ is contained in C.

Now assume that G admits a rainbow cut separating s and t. As s and t are directly connected, every such cut C must contain the edge $\{s, t\}$. Furthermore, by construction C contains exactly one edge from every P_j, say the edge $\{v_{j,\ell_i}, v_{j,\ell_{i+1}}\}$, as no other black edge is part of C. We claim that $M = \{\{v_{\ell_i}, v_{\ell_{i+1}}\} \mid \{v_{j,\ell_i}, v_{j,\ell_{i+1}}\} \in C \text{ for some } j \leq k\}$ is a rainbow matching in P. Obviously, M is rainbow, as C is rainbow. To show that M is indeed a matching, observe that for all $\ell_i \neq \ell_j$ we have $|\ell_i - \ell_j| \geq 2$, that is, M does not contain two (copies of) consecutive edges of P. To see this, assume for the sake of contradiction that there are $i \neq j$ such that $|\ell_i - \ell_j| \leq 1$. Let $j_1 < j_2$ be such that $\{v_{j_1,\ell_i}, v_{j_1,\ell_{i+1}}\}$ and $\{v_{j_2,\ell_j}, v_{j_2,\ell_{j+1}}\}$ are contained in C. By construction, C must also contain $\{u_{j_1,\ell_i}, v_{j_1,\ell_{i+1}}\}$ and can hence not contain $\{u_{j_1,\ell_i}, v_{j_1+1,\ell_i+2}\}$, as they share the same color. This however implies that $\{v_{j_2,\ell_j}, v_{j_2,\ell_{i+1}}\}$ cannot be contained in C, a contradiction. This finishes the proof. ◀

Observe that we can reuse the ideas of the previous proof to show the hardness of the Rainbow Vertex Cut problem. In fact, we can subdivide every edge (and color the subdivision vertex with the same color as the edge), and color every other vertex black. Then all observations translate one-to-one and we obtain the following corollary.

Corollary 7.1.2. *The Rainbow Vertex Cut problem is* NP*-complete on planar graphs.*

An instance of the Weighted Rainbow Vertex/Edge Cut problem consists of an weighted vertex/edge-colored graph (G, φ, w) together with vertices s and t and non-negative integers k and b and the goal is to decide whether there exists a rainbow vertex/edge cut separating s and t of size k collecting weight at most b. We show that the Weighted Rainbow Vertex/Edge Cut problem is fpt with respect to the solution size k given that the weights do not exceed the number of vertices.

Our proof technique employs a heavy toolbox including the *treewidth reduction theorem* [MOR13, Theorem 2.15] and (a variant of) Courcelle's theorem [Cou90]. We refrain from formally introducing these and related notions as we only need a small portion of them. Instead, in addition to the cited works (and references therein), we refer to [CFK+15, Section 7.4] and [FG06, Chapter 11] for an in-depth introduction to treewidth, monadic second-order (MSO) logic and Courcelle's theorem.

Theorem 7.1.3. *The* Weighted Rainbow Vertex/Edge Cut *problem is fixed-parameter tractable with respect to k given that the weight of each vertex is upper-bounded by the number of vertices.*

Proof. We present an fpt-algorithm for the Weighted Rainbow Vertex Cut problem. The tractability of Weighted Rainbow Edge Cut follows by considering the vertex cut version in the line graph of the input graph G, that is, the graph with vertex set $E(G)$ and edge set $\{\{uv, vw\} \mid uv, vw \in E(G)\}$

We follow the approach of [MOR13]. Let (G, φ, w) be a weighted vertex-colored graph with n vertices with the property that $w(v) \leq n$ for every $v \in V(G)$. We first compute a subgraph H of G using the *treewidth reduction theorem* [MOR13]), essentially stating the following. Let C be the set of all vertices of G participating in a minimal vertex cut separating s and t of size at most k for some $s, t \in T$ for fixed subset $T \subseteq V(G)$. For every fixed k, there is a linear-time algorithm that computes a graph H with the following properties:

1. $C \cup T \subseteq V(H)$;
2. For all $s, t \in T$, a set $K \subseteq V(H)$ with $|K| \leq k$ is a minimal vertex cut separating s and t in H if and only if $K \subseteq C \cup T$ and K is a minimal vertex cut separating s and t in G;
3. The treewidth of H is at most $h(k, |T|)$ for some function h; and
4. $H[C \cup T]$ is isomorphic to $G[C \cup T]$, i. e., their vertex and edge sets are equivalent up to names.

We apply the theorem to G with $T = \{s, t\}$ to obtain a subgraph H of G. We inherit the colors and weights from G. Since all minimal vertex cuts of size at most k are preserved in H it is sufficient to search for a minimum weight rainbow vertex cut separating s and t in H of size at most k.

Since the graph H has treewidth at most $h(k, 2)$ we can apply the optimization version of Courcelle's Theorem for graphs of bounded treewidth. For this, observe that we can formulate the existence of a rainbow vertex cut separating s and t as an MSO formula $\varphi(X)$ such that $\varphi(S)$ for a set $S \subseteq V(G)$ is true if and only if S is a rainbow vertex cut. We now apply Courcelle's Theorem in the optimization version presented in [ALS91, CM93]. We remark that the running time in the optimization version of the theorem is usually stated

as being polynomial time for each fixed number of weight functions. This could potentially imply an exponential running time in the given weights. However, it is easily observed that this is not the case and the dependence on each weight function is in fact linear by assumption. We refer to the discussion before Theorem 5.4 in [ALS91] to conclude in our case a running time of $f(k) \cdot n^2$ for some computable function f. ◀

7.1.3 The Sliding Model

In the Vertex Cut Discovery problem under the token sliding model we are given a graph G, vertices $s, t \in V(G)$, a starting configuration $S \subseteq V(G)$ of size k and a non-negative integer b. The goal is to decide whether we can discover a vertex cut separating s and t (starting from S) using at most b token slides. Similarly, in the Edge Cut Discovery problem, we are given a graph G, vertices $s, t \in V(G)$, a starting configuration $S \subseteq E(G)$ of size k and a non-negative integer b. The goal is to decide whether we can discover an edge cut separating s and t (starting from S) using at most b token slides. We denote an instance of Vertex Cut Discovery resp. Edge Cut Discovery by a tuple (G, s, t, S, b).

We prove that Vertex Cut Discovery under the token sliding model is NP-hard, fixed-parameter tractable with respect to parameter k and W[1]-hard with respect to parameter b. We start by proving hardness by a reduction from the Clique problem. Recall that Clique is NP-hard and W[1]-hard with respect to the solution size.

Theorem 7.1.4. *The Vertex Cut Discovery problem under the token sliding model is NP-hard and W[1]-hard with respect to parameter b on 2-degenerate bipartite graphs.*

Proof. We show hardness by a reduction from the Clique problem. The reduction is both a polynomial time reduction as well as an fpt-reduction, showing both claimed results. Let (G, k) be an instance of the Clique problem. We may assume that $k \geq 4$; hence, $\binom{k}{2} > k$.

We construct the following graph H (see Figure 7.2 for an illustration). We add two vertices s and t in H where s has k pendent vertices, which we collect in a set named Z. For every vertex $u \in V(G)$, we add a vertex x_u in H, which we connect with s, and for every $e \in E(G)$ we add a vertex y_e in H, which we connect with t. Let X denote the set of the x_u and Y denote the set of the y_e. For every vertex $u \in V(G)$ and for each edge $e \in E(G)$ incident with u, connect x_u and y_e via a 1-subdivided edge (that is, a path with one interval vertex). Furthermore, we add $\binom{k}{2}$ disjoint paths $P_1, \ldots, P_{\binom{k}{2}}$, each with a single internal vertex, connecting s and t. We denote the internal vertex of P_i by p_i. This completes the construction of H.

Observe that H is 2-degenerate and bipartite. To see that H is 2-degenerate observe that all subdivision vertices (the p_i as well as the subdivision vertices connecting X and Y) have a degree of 2. Their removal yields two stars with centers s and t which are 1-degenerate. Bipartiteness follows from the subdivisions. We define the initial configuration S as $Z \cup Y$ and set the budget to $b = 2k + 2\binom{k}{2}$.

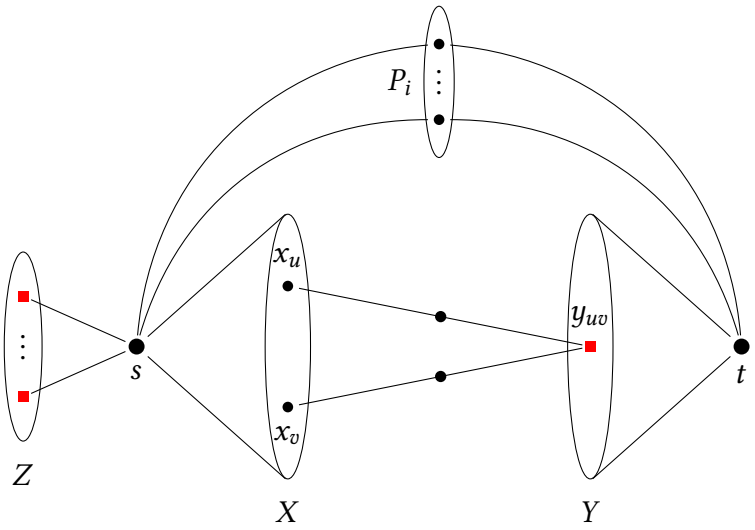


Figure 7.2. Illustration of the hardness reduction for the Vertex Cut Discovery problem.

We claim that G has a clique of size k if and only if (H, s, t, S, b) is a positive instance of the Vertex Cut Discovery problem.

Assume that G has a clique C of size k. In order to separate s and t, we need to cut all (of the $\binom{k}{2}$-many) paths P_i by moving a token on each p_i, and the paths using a vertex from X and a vertex from Y. Using a budget of $2k$, we move the k tokens on the vertices in Z to the k vertices x_u for $u \in C$ by sliding them over s. As C is a clique, this frees $\binom{k}{2}$ tokens in Y, namely those y_e with $x_u \in e$ for $u \in C$ (while the remaining tokens in Y block all other paths connecting X and Y). We move the tokens on these y_e to the subdivision vertices of the P_i by sliding them over t, again for a cost of 2 per token. That is, using $b = 2k + 2\binom{k}{2}$ slides we discover the vertex cut $\{x_u \mid u \in C\} \cup \{y_{uv} \mid u, v \notin C\} \cup \{p_i \mid i \leq \binom{k}{2}\}$ separating s and t, witnessing that (H, s, t, S, b) is a positive instance.

Now assume that (H, s, t, S, b) is a positive instance of the Vertex Cut Discovery problem. Let $C_0 \dots C_\ell$ with $S = C_0$ and $\ell \leq b$ be a configuration sequence in G yielding a vertex cut in G between s and t. Without loss of generality, we assume that ℓ is minimal, i. e., there is no valid solution that can be discovered in less than ℓ steps. In order to cut the P_i, the set C_ℓ must contain p_i for every $i \leq \binom{k}{2}$. Note that the every token in Z as well as every token in Y can reach a p_i in two sliding steps. Hence, we can furthermore assume that a token from Y never moves to a vertex in X. This assumption is justified as C_ℓ contains already $\binom{k}{2}$-many p_i, hence at most k vertices of X are contained in C_ℓ (recall that $\binom{k}{2} \geq k$). Now, instead of moving a vertex from Y to X and moving a vertex from Z to a p_i (each for a cost of 2), we can move a vertex from Z to X and a vertex from Y to a p_i for the same cost instead.

Let z_P be the number of tokens moved from Z to some p_i and $z_X = k - z_P$ be the number of tokens moved from Z to X. Then, $\binom{k}{2} - z_P$ tokens are moved from Y to the remaining p_i. Observe that z_X tokens on X can block at most $\binom{z_X}{2} = \binom{k-z_P}{2}$ paths connecting X and Y. Hence, we have $\binom{k-z_P}{2} \geq \binom{k}{2} - z_P$, as all s-t-paths using vertices from X and Y are cut. As we assume $k \geq 4$, this inequality is only true for $z_P = 0$, that is, no vertex from Z moves to a p_i. Therefore, the (k-many) tokens on Z moved to X and $\binom{k}{2}$ tokens on Y moved to the p_i. Let $y_{uv} \in Y$ be such a vertex that has no token anymore. Now both x_u and x_v must have a token to cut all the paths between s and t passing through y_{uv}. The $\binom{k}{2}$ vertices in Y with no token satisfy the above property if and only if the corresponding edges in G form a k-clique in G, witnessing that (G, k) is a positive instance. ◀

Observe that a small modification of the construction yields the same hardness results for the Edge Cut Discovery problem, as we can keep the same graph H and put the tokens on incident edges instead. To be precise, we put the tokens in Z on the edges connecting Z and s instead, and we put the tokens on Y on the edges connecting Y and t instead. Finally, we set the budget to $b = k + \binom{k}{2}$. Hence, we can move the tokens on edges incident to Z to the edges $\{s, x_u\}$ for $u \in C$ for total cost of k, and the tokens on edges incident to Y to the edges $\{t, p_i\}$ for $i \leq \binom{k}{2}$ for a cost of $\binom{k}{2}$.

Corollary 7.1.5. *The Edge Cut Discovery problem under the token sliding model is* NP*-hard and* W[1]*-hard with respect to parameter b on 2-degenerate bipartite graphs.*

Finally, we use the fixed-parameter tractability of the Weighted Rainbow Vertex/Edge Cut problem to design an fpt-algorithm for Vertex/Edge Cut Discovery problem with respect to parameter k.

Theorem 7.1.6. *The Vertex/Edge Cut Discovery problem under the token sliding model is fixed-parameter tractable with respect to parameter k.*

Proof. We present an fpt-algorithm for the Vertex Cut Discovery problem using the fpt-algorithm for the Weighted Rainbow Vertex Cut problem shown in Theorem 7.1.3 as a subroutine. We note that we can easily adapt the strategy for the Edge Cut Discovery problem using the fpt-algorithm for the Weighted Rainbow Edge Cut problem, respectively.

Our approach is closely related to the *color coding* technique established in [AYZ95]. Let (G, s, t, S, b) be an instance of the Vertex Cut Discovery problem with $n = |V(G)|$. Let C be a color set of k colors. We color the vertices in S arbitrarily with distinct colors from C and the remaining vertices in $V(G) \setminus S$ uniformly at random using colors from C, yielding a vertex coloring $\varphi : V(G) \to C$. Now we define a weight function $w : V(G) \to \mathbb{N}$ such that for each $v \in V(G)$ we have $w(v)$ is equal to the length of a shortest path connecting v to the unique vertex in S with the same color as v. Hence, the weight $w(v)$ denotes the

cost of moving the token on the vertex with the same color as v to v. In particular, $w(u) = 0$ for all $u \in S$ and $w(v) \leq n$ for all $v \in V(G)$.

Consider the instance $(G, s, t, \varphi, w, k, b)$ of the Weighted Rainbow Vertex Cut problem. First we observe that if $(G, s, t, \varphi, w, k, b)$ is a positive instance, then (G, s, t, S, b) is a positive instance of the Vertex Cut Discovery problem. Indeed, let $C \subseteq V(G)$ be a rainbow vertex cut separating s and t collecting weight at most b. Hence, we can find a discovery sequence starting in S and ending in C by moving all tokens in S to the vertex in C with the same color via a shortest path. By the choice of the weight function w, we conclude that the length of the discovery sequence is upper bounded by b, witnessing that (G, s, t, S, b) is a positive instance.

Likewise, we observe that if (G, s, t, S, b) is a positive instance of the Vertex Cut Discovery problem, then $(G, s, t, \varphi, w, k, b)$ is a positive instance of the Weighted Rainbow Vertex Cut with probability at least k^{-k}. This probability is justified by the following observation. Let $C_0 \dots C_\ell$ be a discovery sequence for some $\ell < b$ such that C_ℓ is a vertex cut separating s and t. In particular, given that the vertices in C_ℓ are colored such that each token reaches a vertex of the same color as the vertex the token started on in C_0, we observe that C_ℓ is a vertex cut separating s and t collecting weight at most b with respect to w. As there are k^k ways to color the vertices in C_ℓ with k different colors (recall that $|C_\ell| = k$), and the color of every other vertex is irrelevant, we obtain a success probability of k^{-k}. This probability is tight in the sense that all other (rainbow) colorings of C_ℓ might collect a weight greater than b, hence being too costly.

It remains to show that we can derandomize the construction of $(G, s, t, \varphi, w, k, b)$ in fpt-time in order to utilize the fpt-algorithm for the Weighted Rainbow Vertex Cut problem yielding an fpt-algorithm for the Vertex Cut Discovery problem. However, there are k^{n-k} many possibilities to color the vertices of G with k colors (recall that the k vertices in S have a fixed coloring). Hence, the naïve approach, namely enumerating all these possibilities, does not yield an fpt-algorithm.

To circumvent this problem, we use the derandomization technique in [AYZ95]. Instead of a random coloring φ, we construct a $(n-k, k)$-perfect hash family $\mathcal{F}$ such that for every subset $X \subseteq V(G) \setminus S$ of size k, there is a coloring $c : V(G) \to C \in \mathcal{F}$ with the property every element of X is mapped to a different element in C. The family $\mathcal{F}$ can be constructed deterministically in time $2^{O(k)} \log n$ [AYZ95]. Observe that the weight function w can be computed in time polynomial time, e. g., by the Floyd-Warshall algorithm. Combining these observations with the results in Theorem 7.1.3, we conclude that the Vertex Cut Discovery problem can be solved in time $f(k) \cdot |V(G)|^c$ for some computable function f and a constant c. ◀

7.1.4 Jumping, Adding and Removing, and the Red-Blue Variant

Finally, we study the Red-Blue Vertex/Edge Cut problem and the Vertex Cut Discovery problem under the token jumping model as well as under the token addition/removal model. We begin with the latter problem and show that it remains solvable in polynomial time.

Lemma 7.1.7. *The Vertex/Edge Cut Discovery problem under the token addition/removal model can be solved in polynomial time.*

Proof. We present the strategy for Edge Cut Discovery, the strategy of Vertex Cut Discovery is analogous. Let (G, s, t, S, b) be an instance of the Edge Cut Discovery problem. We may assume that every configuration sequence $C_0 \dots C_\ell$ in G with $C_0 = S$ yielding an edge cut does never remove a token, as every superset of an edge cut is still an edge cut. We consider the graph $G - S$, that is, the graph obtained from G by removing all edges in S. We compute in polynomial time an arbitrary minimum cut C of $G-S$ separating s and t. The size of C is exactly the minimum number of tokens we need to add in order to separate s and t. Hence, if $|C| \leq b$ we conclude that we are dealing with a positive instance. Otherwise we conclude that we are dealing with a negative instance. ◀

Next we show that Red-Blue Vertex/Edge Cut is fpt with respect to parameter k. By Corollary 5.2.4, this result translates to Vertex/Edge Cut Discovery under the token jumping model as well. In fact, we can easily reduce Red-Blue Vertex/Edge Cut to the weighted rainbow version of the problem by giving every blue vertex/edge weight 1 and every red vertex/edge weight 0, and coloring all vertices/edges with distinct colors. The fixed-parameter tractability follows then from Theorem 7.1.3.

Corollary 7.1.8. *The Red-Blue Vertex/Edge Cut problem and the Vertex Cut Discovery problem under the token jumping model are fixed-parameter tractable with respect to k.*

Finally, we note that the reduction in Theorem 7.1.4 showing W[1]-hardness of the Vertex/Edge Cut Discovery problem under the token sliding model with respect to parameter b can be easily adapted to the token jumping model. In fact, we only need to change the budget to $b = k + \binom{k}{2}$ (instead of $2k + 2\binom{k}{2}$ as we only a budget of 1 instead of 2 in order to move a token to its destination). The other observations translate one-to-one. By Corollary 5.2.4 this result also translates to Red-Blue Vertex/Edge Cut.

Corollary 7.1.9. *The Red-Blue Vertex/Edge Cut problem and the Vertex/Edge Cut Discovery problem under the token jumping model are* W[1]*-hard with respect to b.*

8 Conclusion

We have introduced the new framework of *solution discovery via reconfiguration* motivated by the dynamics of real-world applications with the goal to restore healthy system states by executing a sequence of small modifications. We proved many tractability and hardness results concerning solution discovery variants of fundamental graph problems, including classical NP-complete problems as well as problems in P.

We expect further research on this new model capturing the dynamics of real-world situations as well as constraints on the adaptability of solutions. It seems particularly interesting to investigate directed or weighted versions of the studied problems. In future work, we plan to combine our framework with other established frameworks, e. g., dynamic problems, to categorize more of these dynamic real-world applications of decision-making problems. For instance, one can augment the solution discovery framework by allowing two input graphs G and G' for each instance, where G' is obtained from G by a small number of edge/vertex addition/deletions. Moreover, we require the initial configuration S to satisfy certain properties in G. Now the question becomes whether we can transform S to S' using at most b reconfiguration steps such that S' satisfies similar properties in G'. This generalization increases the degrees of freedom in which we can analyze the problems and pushes towards multivariate analyses, where the changes in the graph can now also be part of the parameter.

Another possible avenue to explore is to relax the constraints that we impose on token positions. For instance, we can allow multiple tokens to occupy the same vertex or edge. This, in turn, allows us to decouple the size of the initial configuration from the size of a desired target solution (under the token sliding model and token jumping model). As an example, the Independent Set Discovery problem could be reformulated as follows. We are given a graph G, a starting configuration S, an integer $k \leq |S|$, and a budget b. The goal is now to decide whether we can reach an independent set of size at least k starting from S and using at most b steps.

Furthermore, a challenge is the design of efficient algorithms that can compute approximate solutions with respect to the solution size or with respect to the allowed transformation budget. We note, without proof, that the reduction in Theorem 7.1.4 showing hardness for the Vertex Cut Discovery problem can be adjusted to give a $n^{1-\varepsilon}$-inapproximability of the optimal transformation budget.

Finally, we note that our framework is well-suited for graph problems, but not limited to such problems. As an example, we could consider a discovery variant of the well-known satisfiability problem for propositional logic (SAT) as follows: we are given a formula φ and an arbitrary valuation V of the variables appearing in φ. In particular, φ might be satisfiable but V is *not* a valuation witnessing satisfiability. Then one step could be the flip of the valuation of a single variable, and the question is, given φ, V and budget b, whether at most b such flips suffice in order to discover a valuation such that φ is true under this valuation.

Bibliography

[AKEF+15] Faisal N. Abu-Khzam, Judith Egan, Michael R. Fellows, Frances A. Rosamond, and Peter Shaw. On the parameterized complexity of dynamic problems. *Theoretical Computer Science*, 607:426–434, 2015.

[AL97] Emile Aarts and Jan K. Lenstra. *Local search in combinatorial optimization*. Chichester: Wiley, 1997.

[ALS91] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[AOSS18] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *50th Annual ACM Symposium on the Theory of Computing (STOC 2018)*, pages 815–826. ACM, 2018.

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[BB74] Brenda S. Baker and Ronald V. Book. Reversal-bounded multipushdown machines. *Journal of Computer and System Sciences*, 8(3):315–332, 1974.

[BBC+20] Édouard Bonnet, Nicolas Bousquet, Pierre Charbit, Stéphan Thomassé, and Rémi Watrigant. Parameterized complexity of independent set in H-free graphs. *Algorithmica*, 82(8):2360–2394, 2020.

[BCL20] Xuqing Bai, Renying Chang, and Xueliang Li. More on rainbow disconnection in graphs. *Discussiones Mathematicae Graph Theory*, 42:1185–1204, 2020.

[BDG+23] Pascal Baumann, Flavio D'Alessandro, Moses Ganardi, Oscar Ibarra, Ian McQuillan, Lia Schütze, and Georg Zetzsche. Unboundedness problems for machines with reversal-bounded counters. In *Foundations of Software Science and Computation Structures – 26th International Conference (FOSSACS 2023)*, pages 240–264. Springer, 2023.

[BGLZ24] Pascal Bergsträßer, Moses Ganardi, Anthony W. Lin, and Georg Zetzsche. Ramsey quantifiers in linear arithmetics. *Proceedings of the ACM on Programming Languages*, 8, 2024.

[BHI18] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM Journal on Computing*, 47(3):859–887, 2018.

[BHK18] Hans-Joachim Böckenhauer, Juraj Hromkovic, and Dennis Komm. Reoptimization of hard optimization problems. In *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Methologies and Traditional Applications*, pages 427–454. Chapman and Hall/CRC, 2018.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[BMZ23] Pascal Baumann, Roland Meyer, and Georg Zetzsche. Regular Separability in Büchi VASS. In *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

[BT76] Itshak Borosh and Leon B. Treybig. Bounds on positive integral solutions of linear diophantine equations. *Proceedings of the American Mathematical Society*, 55(2):299–304, 1976.

[Bé79] Étienne Bézout. *Théorie générale des équations algébriques*. Pierres, 1779.

[Bü60] Julius R. Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

[Cad13] Michaël Cadilhac. *Automata with a semilinear constraint*. PhD thesis, Université de Montréal, 2013.

[CCLP17] Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Regular Separability of Parikh Automata. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 117:1–117:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[CFM11] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. On the expressiveness of Parikh automata and related models. In *Third Workshop on Non-Classical Models for Automata and Applications (NCMA 2011)*, volume 282, pages 103–119. Austrian Computer Society, 2011.

[CFM12a] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Affine Parikh automata. *RAIRO Theoretical Informatics and Applications*, 46(4):511–545, 2012.

[CFM12b] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Bounded Parikh automata. *International Journal of Foundations of Computer Science*, 23(8):1691–1710, 2012.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. The MIT Press, 1999.

[CH16] Dmitry Chistikov and Christoph Haase. The Taming of the Semi-Linear Set. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 128:1–128:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

[CHVB18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of Model Checking*. Springer, 1st edition, 2018.

[CJK13] Chen C. Chang and Howard J. Keisler. *Model Theory: Third Edition*. Dover Books on Mathematics. Dover Publications, 2013.

[CM93] Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1-2):49–82, 1993.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC 1971)*, pages 151–158. Association for Computing Machinery, 1971.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[DDF+14] Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Polynomial-time algorithm for sliding tokens on trees. In *16th International Society for Augmentative and Alternative Communication (ISAAC 2014)*, pages 389–400. Springer, 2014.

[DF92] Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness. In *Currents in Research*, volume 87, pages 191–225, 1992.

[DF95] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[DFT19] Luc Dartois, Emmanuel Filiot, and Jean-Marc Talbot. Two-way Parikh automata with a visibly pushdown stack. In *Foundations of Software Science and Computation Structures – 22nd International Conference (FOSSACS 2019)*, volume 11425 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2019.

[Dic13] Leonard E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913.

[DK09] Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In *29th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009)*, pages 157–168. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009.

[DS14] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *46th Annual ACM Symposium on the Theory of Computing (STOC 2014)*, pages 624–633. ACM, 2014.

[EGJ+23] Enzo Erlich, Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. History-Deterministic Parikh Automata. In *34th International Conference on Concurrency Theory (CONCUR 2023)*, volume 279 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

[EK72] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[FGM19] Emmanuel Filiot, Shibashis Guha, and Nicolas Mazzocchi. Two-way Parikh automata. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*, volume 150 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[FGM+23] Michael R. Fellows, Mario Grobler, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Frances A. Rosamond, Daniel Schmand, and Sebastian Siebertz. On Solution Discovery via Reconfiguration. In *26th European Conference on Artificial Intelligence (ECAI 2023)*, volume 372 of *FAIA*, pages 700–707, 2023.

[FL15] Diego Figueira and Leonid Libkin. Path logics for querying graphs: Combining expressiveness and efficiency. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2015)*, pages 329–340, 2015.

[FS02] Henning Fernau and Ralf Stiebe. Sequential grammars and automata with valences. *Theoretical Computer Science*, 276(1):377–405, 2002.

[FS08] Henning Fernau and Ralf Stiebe. Blind counter automata on omega-words. *Fundamenta Informaticae*, 83:51–64, 2008.

[GH06] Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33(9):2547–2562, 2006.

[GJ79] Michael R. Garey. and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1st edition, 1979.

[GJLZ22] Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. Parikh Automata over Infinite Words. In *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022)*, volume 250 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

[GL02] Dimitra Giannakopoulou and Flavio Lerda. From states to transitions: Improving translation of LTL formulae to Büchi automata. In *Formal Techniques for (Networked and) Distributed Systems*, 2002.

[GMM+23] Mario Grobler, Stephanie Maaz, Nicole Megow, Amer E. Mouawad, Vijayaragunathan Ramamoorthi, Daniel Schmand, and Sebastian Siebertz. Solution discovery via reconfiguration for problems in P, 2023.

[GNS20] Guilherme Gomes, Sérgio H. Nogueira, and Vinicius F. dos Santos. Some results on vertex separator reconfiguration. *arXiv preprint arXiv:2004.10873*, 2020.

[Gre78] Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3):311–324, 1978.

[GS53] David Gale and Frank M. Stewart. Infinite games with perfect information. In *Contributions to the Theory of Games (AM-28), Volume II*, pages 245–266. Princeton University Press, 1953.

[GS64] Seymour Ginsburg and Edwin H. Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.

[GS24] Mario Grobler and Sebastian Siebertz. Deterministic parikh automata on infinite words, 2024.

[GSS23] Mario Grobler, Leif Sabellek, and Sebastian Siebertz. Parikh automata on infinite words. *arXiv preprint arXiv:2301.08969*, 2023.

[GSS24] Mario Grobler, Leif Sabellek, and Sebastian Siebertz. Remarks on Parikh-Recognizable Omega-languages. In *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke. *Automata logics, and infinite games: a guide to current research*. Springer, 2002.

[Haa14] Christoph Haase. Subclasses of presburger arithmetic and the weak exp hierarchy. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL 2014) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2014)*. Association for Computing Machinery, 2014.

[Haa18] Christoph Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018.

[Heu13] Jan van den Heuvel. The complexity of change. *Surveys in Combinatorics*, 409(2013):127–160, 2013.

[HH14] Christoph Haase and Simon Halfon. *Integer Vector Addition Systems with States*, pages 112–124. Springer, 2014.

[Hig52] Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of The London Mathematical Society*, pages 326–336, 1952.

[HIM+16] Arash Haddadan, Takehiro Ito, Amer E. Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theoretical Computer Science*, 651:37–49, 2016.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2006.

[HN13] Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494:86–98, 2013.

[HNW07] Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Techniques for Practical Fixed-Parameter Algorithms. *The Computer Journal*, 51(1):7–25, 2007.

[Hoo02] Hendrik J. Hoogeboom. Context-free valence grammars - revisited. In *Developments in Language Theory*, pages 293–303. Springer, 2002.

[HPQ17] Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. *SIAM Journal on Computing*, 46(6):1712–1744, 2017.

[Huy80] Dung T. Huynh. The complexity of semilinear sets. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming (ICALP 1980)*, pages 324–337. Springer, 1980.

[Huy86] Dung T. Huynh. A Simple Proof for the Σ_2^{P} Upper Bound of the Inequivalence Problem for Semilinear Sets. *Journal of Information Processing and Cybernetics*, 22(4):147–156, 1986.

[HZ21] Christoph Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2019)*. IEEE Press, 2021.

[Iba78] Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116––133, 1978.

[IDH+11] Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011.

[Jaf78] Jeffrey Jaffe. A necessary and sufficient pumping lemma for regular languages. *SIGACT News*, 10(2):48––49, 1978.

[JK03] Matthias Jantzen and Alexy Kurganskyy. Refining the hierarchy of blind multicounter languages and twist-closed trios. *Information and Computation*, 185(2):159–181, 2003.

[Joh73] David S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing (STOC 1973)*, pages 38–49. ACM, 1973.

[Kar72] Richard Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[Kar04] Wong Karianto. Parikh automata with pushdown stack. *Diplomarbeit, RWTH Aachen*, 2004.

[KMM12] Marcin Kaminski, Paul Medvedev, and Martin Milanic. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.

[Koz97] Dexter C. Kozen. *Automata and Computability*. Springer, 1st edition, 1997.

[KR03a] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-ε. In *Computational Complexity Conference (CCC 2003)*, page 379. IEEE Computer Society, 2003.

[KR03b] Felix Klaedtke and Harald Ruess. Monadic second-order logics with cardinalities. In *Automata, Languages and Programming*, pages 681–696. Springer, 2003.

[KRS18] Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. *Transactions on Algorithms*, 15(2):1–19, 2018.

[KST18] Ramaswamy Krithika, Abhishek Sahu, and Prafullkumar Tale. Dynamic parameterized problems. *Algorithmica*, 80(9):2637–2655, 2018.

[Kur87] Robert P. Kurshan. Complementing deterministic büchi automata in polynomial time. *Journal of Computer and System Sciences*, 35(1):59–71, 1987.

[Lan69] Lawrence H. Landweber. Decision problems for ω-automata. *Mathematical systems theory*, 3:376–384, 1969.

[Lat79] Michel Latteux. Cônes rationnels commutatifs. *Journal of Computer and System Sciences*, 18(3):307–333, 1979.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[Lic82] David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[Lov75] László Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975.

[LP14] Van Bang Le and Florian Pfender. Complexity results for rainbow matchings. *Theoretical Computer Science*, 524:27–33, 2014.

[LPS+20] Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Covering small independent sets and separators with applications to parameterized algorithms. *ACM Transactions on Algorithms*, 16(3):32:1–32:31, 2020.

[McN66] Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and control*, 9(5):521–530, 1966.

[Men27] Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, 1967.

[MNR+17] Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017.

[Moh01] Bojan Mohar. Face covers and the genus problem for apex graphs. *Journal of Combinatorial Theory, Series B*, 82(1):102–117, 2001.

[MOR13] Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms (TALG)*, 9(4):1–35, 2013.

[MS01] Victor Mitrana and Ralf Stiebe. Extended finite automata over groups. *Discrete Applied Mathematics*, 108(3):287–300, 2001.

[Nis18] Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

[OR10] Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *42nd Annual ACM Symposium on the Theory of Computing (STOC 2010)*, pages 457–464. ACM, 2010.

[Par66] Rohit J. Parikh. On context-free languages. *Journal of the ACM (JACM)*, 13(4):570–581, 1966.

[Sin09] Arindama Singh. *Elements of Computation Theory*. Springer, 1st edition, 2009.

[Sip13] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, third edition, 2013.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing (STOC 1973)*, pages 1–9. Association for Computing Machinery, 1973.

[Sto76] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[Tho91] Wolfgang Thomas. *Automata on Infinite Objects*, page 133–191. MIT Press, 1991.

[To10] Anthony W. To. Parikh images of regular languages: Complexity and applications, 2010.

[TV19] Jan A. Telle and Yngve Villanger. Fpt algorithms for domination in sparse graphs and beyond. *Theoretical Computer Science*, 770:62–68, 2019.

[VSS05] Kumar N. Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational horn clauses. In *Proceedings of the 20th International Conference on Automated Deduction (CADE 2020)*, pages 337–352. Springer, 2005.

[vzGS78] Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1):155–158, 1978.

[Wro18] Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *Journal of Computer and System Sciences*, 93:1–10, 2018.

[Zet13] Georg Zetzsche. Silent transitions in automata with storage. In *Automata, Languages, and Programming*, pages 434–445. Springer, 2013.

[Zuc06] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *38th Annual ACM Symposium on the Theory of Computing (STOC 2006)*, pages 681–690. ACM, 2006.