Connections between arithmetic theories and infinite-state systems

Andrei Draghici
St Catherine's College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Hilary 2025

Acknowledgements

First and foremost, I would like to thank my advisor, Christoph Haase, for many things. He was one of my favourite teachers during my undergrad studies at Oxford and also the one responsible for infecting me with this strange passion for arithmetic theories. He was also the reason I decided to pursue a DPhil in the first place and I am so glad that I did. I truly couldn't have asked for a better advisor–this journey has been both rewarding and a lot of fun, thanks to him.

I would also like to thank my co-advisor, Marta Kwiatkowska, for her support and guidance throughout this journey.

A big thank you goes to my co-authors–Florin Manea, Andrew Ryzhikov, Radosław Piórkowski–for all the fun and valuable meetings we had, and also the less fun crises we had before every conference deadline.

Special thanks go to Georg Gottlob, my Master's advisor, who was one of the first mentors in my academic career. His advice on identifying meaningful research questions and on presenting proofs and results with clarity has guided me throughout my DPhil.

I am also grateful to all my undergraduate professors, with special thanks to Gavin Lowe and Karel Hruda, who inspired me to approach problems deeply and appreciate their intrinsic beauty.

Outside the academic world, there are many people who shaped my journey and, in many ways, made this thesis possible. I had wonderful teachers throughout school and high school, and I am immensely thankful to my family for their unwavering support at every step. I'm also grateful to my close group of friends—who I truly consider one of my greatest accomplishments—with a special mention to my housemate Alex, who made life in Oxford even more enjoyable.

Abstract

Infinite-state systems are crucial for modelling the behaviour of computer programs, concurrent processes, and reasoning about formal languages. One fundamental area of research in theoretical computer science is the study of the computational complexity of the decision problems associated with these models. Similarly, arithmetic theories, such as Presburger arithmetic, are used to model and check properties of programs such as loop invariants, and termination conditions. In 1960, Büchi managed to relate the two concepts via the notion of automaticity, the idea of using a DFA to model the solution set of a formula of Presburger arithmetic. In this thesis, we provide new results in both areas, by leveraging the relationship between them.

First, we study extensions of Semënov arithmetic, the first-order theory of the structure $\langle \mathbb{N}, +, 2^x \rangle$. It is well-known that this theory becomes undecidable when extended with regular predicates over tuples of number strings, such as the Büchi V_2-predicate. We therefore restrict ourself to the existential fragment, and prove that this theory is decidable in EXPSPACE when extended with arbitrary regular predicates, by establishing a connection to a new model of computation, a type of vector addition system with states (VASS), where we allow for affine updates on the counters.

Next, we look at the reachability problem for VASS. Existing work primarily considers the case in which both the VASS as well as the initial and target configurations are part of the input. Here, we investigate the setting in which the VASS and target configurations are fixed, and only the initial configuration is variable. We show that fixed VASS fully express arithmetic with counting on initial segments of the natural numbers. It follows that there is a very weak reduction from any fixed such number-theoretic predicate to reachability in fixed VASS where configurations are presented in unary.

Finally, we prove that the boundedness problem for cost register automata (CRA), which asks if there exists a natural number N such that for every word, the value of the CRA on that word does not exceed N is decidable for copyless linear CRAs with at most two registers over the tropical semiring.

Contents

Chapter 1

Introduction

This chapter offers an informal introduction to the key concepts and challenges addressed in this thesis, establishing the motivation and the scope of our work. We begin by discussing the motivation. Verifying the correctness of computer systems is a well-recognized challenge–one that is both essential and notoriously difficult to achieve. There are countless examples of instances where human lives depend on the flawless operation of systems, such as in aviation control, autonomous vehicles, and medical devices. The most popular method of solving this challenge in industry is the use of rigorous testing. However, the downside of this method is that it cannot provide any type of guarantee for the well-functioning of the systems. Given the critical nature of these systems, we must go beyond conventional testing and focus on formal methods, which offer a mathematical approach to system verification.

The aim of formal verification is to use mathematical methods to prove or disprove the correctness of a computer system with respect to a given specification. Since formal methods cannot be applied to concrete systems, the first step is to find an appropriate abstract model that approximates well the object we wish to verify. Although the field of software formal verification is relatively new, the idea of using mathematical objects to model the world is not. The first recorded instance of humans modelling the real world with mathematics is often attributed to the Babylonians around 1800 BCE, who used mathematics to model and predict astronomical phenomena [40]. A closer example for our parallel is the use of mathematical equations in order to model the motion of objects in classical mechanics. Just as the equations of motion allow us to predict that a free-falling object will not exceed a certain speed before hitting the ground, abstract models can enable us to prove that a computer system will respect a given property.

The mathematical structures used for this modelling are not only practical, but also intrinsically elegant. This dual nature might be one reason for which the field

of mathematics developed independently of their use for physical modelling. One can study the properties of systems of equations without thinking of their practical application. Similarly, in Computer Science, we study the properties of abstract models such as Deterministic Finite Automata, Vector Addition Systems, Cost Register Automata without always thinking of their connection with the computer systems that we can model with them. This is a worthwhile task in itself and one that in the past brought numerous practical advances without having this explicit goal. In this thesis, we focus on abstract models that fall under the broad category of Infinite-State Systems. To explore their properties, we draw parallels between them and arithmetic theories, which are detailed the following sections. This connection serves as a foundation for discovering new results on both the models and the arithmetic theories, representing a central contribution of this thesis. The next section provides an informal overview of the abstract models that we study, highlighting both their shared foundations and their unique characteristics, together with their connection to arithmetic theories.

1.1 Formal Models and Arithmetic Theories

The classical example of modelling a computer system given to first year computer science students is that of modelling a vending machine with the help of a deterministic finite automata (DFA). A vending machine receives limited input from the outside world and based on this input and its internal state, it moves from one state to another. Similarly, a DFA is a finite-state model that receives inputs from a limited alphabet and moves from one state to another based on the input and the previous active state. Thus, one can answer questions such as "Does the vending machine ever deliver coffee?" by investigating if the DFA can reach the "coffee delivery" state or not. This can be seen as an *emptiness* or a *reachability* query on the DFA. The DFA is one of the simplest models out there and fortunately, we have efficient algorithms for solving the emptiness problem (and many others) for it. Because of this, the model is widely used in lexical analysis for programming languages, where they efficiently tokenise the input by recognising patterns such as keywords and identifiers. DFAs are also applied in network protocol design and text processing, where their ability to match regular patterns ensures correctness and efficiency.

Apart from the practical applications, the study of DFAs also lead to developments in the area of arithmetic theories [19,46]. Arithmetic theories provide a formal framework to study the properties of numbers and operations. They also have multiple

applications in the field of formal verification. Among these, Presburger Arithmetic, the first-order theory of the integers with addition, equality and the standard axioms of arithmetic is highly relevant since it was proven to be decidable by Presburger [74] in 1929 via a quantifier elimination procedure. That means that given a sentence in the structure $\langle \mathbb{N}, 0, 1, +, < \rangle$, there exists and algorithm that can decide if the sentence is provable from the axioms of Presburger arithmetic or not. In 1960, Büchi developed a new decision procedure for Presburger arithmetic that relied on associating a DFA with every existential formula in Presburger arithmetic, such that the formula is unsatisfiable if and only if the language of the DFA is empty [19]. This new method of generating the solutions of a first-order formula of an arithmetic theory by a DFA provided a characterisation of the expressivity of Presburger arithmetic and also gave rise to the more general concept of *automatic structures* [46]. Consider the following trivial example. Let $f = \exists x.x > 3$ be an existential formula of Presburger arithmetic. Clearly, this formula is true and the set of solutions is $S = \{x \in \mathbb{N} | x > 3\}$. Now, consider the following DFA.

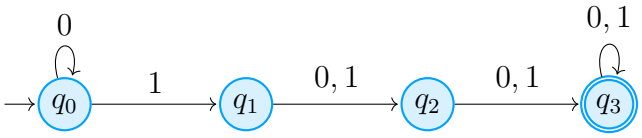


Figure 1.1: A DFA accepting all integers greater than 3.

The language the DFA is composed by all the binary strings representing integers greater than 3, where the most significant bit is at the left of the word. Thus, the language of the DFA is non-empty if and only if the formula f is true.

Even though DFAs can model formulas of Presburger arithmetic, their expressive power is still quite limited, mainly due to their finiteness. Thus, a simple way to make them more expressive is to add to them a finite number of counters ranging over the non-negative integers. Since the domain of the added counters is infinite, the state-space of the model also becomes infinite and we can no longer solve reachability or emptiness queries by exploring the entire state-space. These problems become indeed a lot harder, and without limiting the way in which the counters are updated, the problems become undecidable. Thus, in the case of Vector addition systems with states (VASS), we only allow constant updates for the counters of the type $x \leftarrow x + b, b \in \mathbb{N}$.

The VASS reachability problem has been one of the most intriguing problems in theoretical computer science and studied for more than fifty years. In the 1970s,

Lipton showed this problem EXPSPACE-hard [63]. Ever since the 1980s [56, 60, 66], the reachability problem has been known to be decidable, albeit with non-elementary complexity. This wide gap between the EXPSPACE lower bound and a non-elementary upper bound persisted for many years, until a recent series of papers established various non-elementary lower bounds [25, 61], and resulted in matching a recently established upper bound [27, 59], showing the VASS reachability problem Ackermann-complete. We leverage the techniques developed in these works (mainly the idea of delaying the zero-test) in order to conclude that VASS can model formulas of the first-order theory of integers with addition, equality, and multiplication, when we restrict ourselves to initial segments of the integers. This fragment of $\langle \mathbb{Z}, +, \times \rangle$ is called *rudimentary arithmetic*. By using this connection, we manage to get new insights into the expressive power of VASS and also derive new complexity results for the reachability problem when the input VASS is considered fixed.

Allowing more powerful update functions for the counters gives rise to new models. Thus, by allowing the counters of VASS to also have affine updates of the type $x \leftarrow ax + b$ when taking a transition, we get a new model which we call labelled affine VASS (LAVASS). We design this new model in order to be able to recognise the solution sets of existential formulas of an extension of the structure $\langle \mathbb{N}, 0, 1, +, 2^x \rangle$, where 2^x is the function mapping a natural number n to 2^n. Decidability of the first-order theory of this structure was first shown by Semënov in a more general framework using an automata-theoretic approach [83], and we henceforth call this theory *Semënov arithmetic*. Unlike its substructure Presburger arithmetic (obtained by dropping the function 2^x), the solution set of an existential formula of Semënov arithmetic cannot be recognised by a DFA. The decidability of Semënov arithmetic is fragile with respect to extensions of the structure. For instance, it is not difficult to see that extending Semënov arithmetic with the Büchi predicate $V_2(x, y)$, where $V_2(x, y)$ holds whenever x is the largest power of two dividing y without remainder, results in an undecidable first-order theory, see e.g. [73].

The reachability problem for LAVASS becomes undecidable even in the presence of two counters [75], so we impose several structural restrictions to the model that make the reachable problem solvable, while retaining its ability to model formulas of Semënov arithmetic. We split the counters into pair of counters in which the first counter can only be updated by additive updates $x \leftarrow x + 1$ and the second one can only be updated via affine updates $x \leftarrow 2x$ and $x \leftarrow 2x + 1$. A configuration is accepting if and only if we are in an accepting control state and the values of every pair of counters are equal. For this new model, we give an EXPSPACE procedure

for deciding emptiness that relies on a counter elimination procedure in which we successively encode the counters in an abstract finite state-space while preserving equi-non-emptiness.

Another powerful model that we study in this thesis is that of Cost Register Automata (CRA), introduced by Alur et al. [5], in which we allow the counters to interact with each other. The update functions are over the tropical semiring $(\mathbb{Z} \cup \{\infty\}, \min, +)$, and we refer to the counters as registers to emphasize the fact that they can exchange values between them, e.g. $x \leftarrow \min\{y+3, z+1\}, y \leftarrow x+2$. In contrast to the previous models, CRAs are designed to compute numerical functions over input strings, making them particularly well-suited for quantitative analysis. Also, since we regard CRAs as functions, we are no longer interested in reachability queries, but rather in problems such as *equivalence* or *boundedness*. The latter asks, given a CRA, whether there exists a natural number N such that the output of the CRA on every input is smaller than N. For general CRAs, the boundedness problem is undecidable. Furthermore, it is shown in [3] that the problem remains undecidable even for the restricted class of *copyless* CRAs. In this class, no register is allowed to be used twice in any update function. We focus on the class of *linear* copyless CRAs, where we further more require the register updates to also be linear. This still remains a very powerful subclass, so we introduce further restrictions. First, we limit the number of registers to only two. This is well-founded since for many problems the case of automata with two counters or registers often turns out to be already difficult enough. For example, most decision problems are undecidable for two-counter automata [69]. For vector addition systems with states, decidability of the reachability problem in the two-counter case was established in a seminal paper [50] several years before the proof that it is decidable for arbitrary number of counters was found [65], and it took over 20 more years to establish the precise computational complexity of the two-counter case [15]. Secondly, we study the class of CRAs where the minimum operation only occurs in the output function. Here, the model turns out to be very similar to a $\mathbb{Z}$-VASS where we also allow permutations of counters.

We do not manage to find a suitable connection between this model and a an arithmetic theory, mainly due to the highly non-linear behaviour of iterated minimums. However, the better understanding of the properties of the restricted subclasses of CRAs might lead to finding new decidable extension of Presburger arithmetic.

1.2 Structure of this Thesis

Here, we present the organisation of the thesis. In Chapter 2, we introduce basic notation, definitions, and present some fundamental results that we use. The aim is to provide a common background for all the future results presented in the thesis. Although this thesis is self-contained, the fields covered have produced a very rich body of literature, so we expect the reader of this thesis to have some familiarity with the basic concepts and standards from computational logic (with an emphasis on arithmetic theories), formal languages, and complexity.

In Chapter 3, we provide a new form of automaticity to an extension of Semënov arithmetic with the use of a new model called LAVASS. The extension that we study is the existential theory of the structure $\langle \mathbb{N}, 0, 1, +, 2^x, \{R_i\}_{i\geq 0}\rangle$, where $R_0, R_1, \ldots$ is an enumeration of all regular languages over the alphabets $\{0,1\}^d$, which we call *generalised Semënov arithmetic*. First, we manage to prove that the solution set of an existential formula of generalised Semënov arithmetic can be described by the language of LAVASS. This already gives us a new form of automaticity for the theory. We believe that this approach is quite general and fruitful, so we are hopeful that it can be applied to other arithmetic theories as well. One argument in favour of this approach is that after we established the automaticity of the initial structure of interest $\langle \mathbb{N}, 0, 1, +, 2^x, <, V_2(x,y)\rangle$, where $V_2(n)$ represents the function taking n to the largest power of 2 divides n, we immediately managed to extend our results to the more expressive structure $\langle \mathbb{N}, 0, 1, +, 2^x, \{R_i\}_{i\geq 0}\rangle$ by using previous results on the automaticity of Büchi arithmetic. Next, by studying the properties of the structure of LAVASS that model formulas of generalised Semënov arithmetic, we find a restricted class of LAVASS that maintains this modelling power. Finally, the main result of the chapter is an EXPSPACE procedure that decides the emptiness problem for this restricted class. This immediately allows us to conclude that the problem of deciding generalised Semënov arithmetic is in EXPSPACE.

In Chapter 4, we adopt a similar approach to that used in Chapter 3, but in the opposite direction. Specifically, in Chapter 3, we begin with an arithmetic of interest and establish a connection to a new formal model to address questions about the arithmetic. In this chapter, however, we focus on analysing a well-established formal model (VASS) and developing a connection to arithmetic theories, aiming to derive new results for the model.

Lastly, in Chapter 5, we study CRAs and the boundedness problem for them. We look at linear copyless CRAs over the tropical semiring $(\mathbb{Z} \cup \{\infty\}, \min, +)$. The

problem is known to be undecidable both for copyless CRAs and for linear CRAs. The main result of this chapter is proving that boundedness is decidable for linear copyless CRAs with at most two registers. Namely, we show that it is NL-complete if the numbers in the updates are presented in unary, and is in coNP if they are presented in binary. Functions computed by CRAs, even when they have only two registers, are highly non-linear and complex, mainly due to the presence of nested minimum operations. We illustrate this by providing in Section 5.5 a series of CRAs with very long shortest runs outputting a given value.

Each chapter starts with an overview of related work from the literature. These discussions highlight the research that aligns with the main themes of this thesis, situating our contributions within the broader context of the field and identifying connections to existing studies. The chapters then conclude with a discussion of the results obtained, and potential directions for future research.

1.3 Joint Work and Copyright Notice

The results presented in this thesis are mainly based on peer-reviewed publications that have been co-authored with a number of collaborators. The publications are the result of numerous meetings and discussions between the author and his collaborators.

The results in Chapter 3 have been published in the proceedings of the 41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12–14, 2024, Clermont-Ferrand, France as Andrei Draghici, Christoph Haase, Florin Manea, Semënov Arithmetic, Affine VASS, and String Constraints, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pages 29:1–29:19. Florin posed the question of deciding generalised Semënov due to the connection between this theory and a theory of string constraints which is detailed in the published article. While working on this question I came up with the idea of using a special type of automata in order to decide the theory. After this, I had multiple meetings with my advisor Christoph in which he offered numerous advice on how to push through with the general idea of the proof while I was working out the technical details.

The results of Chapter 4 have been published in Foundations of Software Science and Computation Structures - 27th International Conference, FoSSaCS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6–11, 2024, Proceedings, Part II, as Andrei Draghici, Christoph Haase, and Andrew Ryzhikov, Reachability in Fixed VASS: Expressiveness and Lower Bounds, Lecture Notes in Computer Science, Springer,

pages, 185–205. Christoph came to me with the idea of relating VASS and Short Presburger arithmetic. After this, I started working on devising the right components that could model the functions and predicates of the theory. This gave us the first hardness result for binary inputs. Next, Andrew joined the project and after multiple discussion together with Christoph we came up with the concept of unary reductions in order to deal with unary inputs. Finally, during the talks we discovered that there is more direct route via a series of reduction to a PSPACE bound for the binary inputs.

Lastly, the results in Chapter 5 have been published in the 33rd EACSL Annual Conference on Computer Science Logic (CSL 2025), as Andrei Draghici, Radosław Piórkowski, and Andrew Ryzhikov, Boundedness of Cost Register Automata over the Integer Min-Plus Semiring, Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, pages, 20:1–20:23. The project started after Andrew taught me about CRAs and the boundedness problem. After some friendly chats in the department and some independent work, I came up with a very rough proof for the two register case. Radosław helped us to refine and also present the proof in a nicer way. We also provide special thanks to Radosław for graciously allowing the inclusion of his beautiful illustrations in this chapter, enhancing the readability and appeal of the chapter. Finally, the example of the state-less CRA came up during multiple discussions among the three of us together with the reductions for CRAs with minimum at the end, for which I worked out the details for the lower bounds.

Chapter 2

Preliminaries

In this chapter, we introduce general notation, definitions, and recall some fundamental results that we use in this thesis. We start with some basic notation on integers and formal languages. Next, we describe the arithmetic theories that we study and describe their connection to formal languages via the notion of automaticity. Finally, we define the formal models that we study and observe that they are all infinite-state systems because of the combination of automata and integer counters.

2.1 General Notation

By $\mathbb{R}, \mathbb{Q}, \mathbb{Z}$ we denote the set of *reals*, *rationals*, and *integers* respectively. We denote by $\mathbb{R}_{\geq 0} := \{r \in \mathbb{R} : r \geq 0\}$, by $\mathbb{Q}_{\geq 0} := \{q \in \mathbb{Q} : q \geq 0\}$, and by $\mathbb{Z}_{\geq 0} := \{z \in \mathbb{Z} : z \geq 0\}$. We also denote by $\mathbb{N} := \mathbb{Z}_{\geq 0}$ the set of *natural numbers*. For $N \in \mathbb{N}$ we write $\underline{N}$ to denote the set $\{0, \ldots, N\}$. By $[n, m]$ we define the set of integers between n and m: $[n, m] := \{z \in \mathbb{Z} : n \leq z \leq m\}$. For any $z \in \mathbb{Z}$, we denote by $|z|$ the *absolute value* of z. For a set M, 2^M is the *power set* of M and M^k is the k-fold *Cartesian product* of M with itself for any integer $k \geq 1$. We overload the $|\cdot|$ operator by denoting the size of a finite set M by $|M|$. For a set $M \subseteq \mathbb{Z}$ with a minimum or maximal element, we denote by $\min M$ and $\max M$ its minimum and maximal element, respectively. For all $r \in \mathbb{R}$, we define the *floor function* $\lfloor r \rfloor := \max\{z \in \mathbb{Z} : z \leq r\}$ and the *ceiling function* $\lceil r \rceil := \min\{z \in \mathbb{Z} : z \geq r\}$. For a vector $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and a real number $p \geq 1$, we denote by $\|\boldsymbol{x}\|_p := (\sum_{i=1}^{n} |x_i|^p)^{1/p}$ its *p-norm*. For matrix $A \in \mathbb{Z}^{m \times n}$, we denote by $\|A\|_{1,\infty} := \max_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|$ the $(1, \infty)$-norm.

In the context of formal languages, we mainly use the notation and concepts found in Sipser's book [85]. Let Σ be a finite set of symbols which we sometimes call an alphabet. The *Kleene star* is denoted by $\cdot^*$ and thus, we denote by Σ^* the set of all finite-length strings over symbols in Σ including the *empty string* which we denote

by ϵ. For any two strings $u, v \in \Sigma^*$, we denote by uv the concatenation of u and v. A subset $L \subseteq \Sigma^*$ is called a *language* over Σ. Moreover, any language L that is accepted by a *deterministic finite automaton (DFA)* is called *regular*. For a regular language L, we denote by $L^C := \Sigma^* \setminus L$, the complement of L. When speaking of underlying digraphs of NFAs, we use standard terms like simple cycle, and reachability. We emphasise that by a simple cycle we mean a cycle that does not visit the same vertex more than once, except for its first and last vertex. General cycles do not have this restriction.

For $\Sigma = \{0, 1\}$, any string from $\Sigma^* \setminus \epsilon$ has an interpretation as a binary encoding of a natural number, possibly with an arbitrary number of leading zeros. Conversely, any natural number in $\mathbb{N}$ can be converted into its bit representation as a string in Σ^*. Finally, by considering strings over $(\Sigma^k)^*$ for $k \geq 1$, we can represent k-tuples of natural numbers as strings over Σ^k, and *vice versa*. Formally, given $u = \boldsymbol{u}_n \boldsymbol{u}_{n-1} \dots \boldsymbol{u}_0 \in (\Sigma^k)^*$, we define the tuple of natural numbers corresponding to u in *most-significant digit first (msd)* notation as

$$[\![u]\!] := \sum_{i=0}^{n} 2^i \cdot \boldsymbol{u}_i \,.$$

Note that $[\![\cdot]\!]$ is surjective but not injective. We lift the definition of $[\![\cdot]\!]$ to sets in the natural way.

2.2 Arithmetic Theories

In this section, we define the notion of *automatic structures* which enables us to exploit the connection between languages and integers discussed in the previous section. We also present multiple first-order theories on integers and recall some decidability results. We assume some familiarity with the basic definitions of first-order logic that can be found for e.g., in [78], [11]. Given a σ-structure $\mathcal{A}$, we denote by $\mathrm{Th}(\mathcal{A})$ the *theory of* $\mathcal{A}$ which is the set of all closed formulas that are *modelled* by $\mathcal{A}$, i.e., $\mathrm{Th}(\mathcal{A}) := \{F : \mathcal{A} \models F\}$. In this thesis, we assume all σ-structures to be *relational*, i.e., σ contains only relation symbols. Notice that this comes with no loss of generality since any k-ary function symbol f can be turned into a relation R_f such that $R_f = \{(a_1, \dots, a_k, b) : f(a_1, \dots, a_k) = b\}$. Also, a constant symbol can be viewed as a 0-ary function symbol. A theory $\mathcal{T}$ is said to be *decidable* if there exists an algorithm that, given a sentence F, decides whether or not $F \in \mathcal{T}$.

2.2.1 Automatic Structures

Consider a relational σ-structure $\mathcal{A}$ such that the universe of $\mathcal{A}$ is the set of natural numbers. Let $\Sigma = \{0, 1\}$ and recall the correspondence between strings over $(\Sigma^k)^*$ and k-tuples of natural numbers from the previous section. For a quantifier-free formula Φ with free variables $\boldsymbol{x}$, we denote by $[\![\Phi]\!] := \left\{\boldsymbol{x} \in \mathbb{N}^k : \Phi(\boldsymbol{x})\right\}$ the set of all satisfying assignments of Φ. Let $\mathcal{U}_\mathcal{A}$ be the universe of the σ-structure $\mathcal{A}$. A k-ary relation $R \subseteq \mathcal{U}_\mathcal{A}^k$ of $\mathcal{A}$ is *automatic* if the language $L_R := \left\{u : u \in (\Sigma^k)^*, [\![u]\!] \in R\right\}$ is regular. For example, the relation $R = \{(n, m) : n = m\}$ is automatic because it can be expressed as the following regular expression $\left\{\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\right\}^*$. A relational structure $\mathcal{A} = (\mathbb{N}, R_1, \ldots, R_m)$ is *automatic* if all the relations $R_1, \ldots, R_m$ are automatic, and a structure $\mathcal{A}$ has an *automatic representation* if $\mathcal{A}$ is isomorphic to an automatic structure. Note that in general, one does not need to restrict the concept of automatic structures to structures where the universe is the set of natural numbers, however, this will suffice for our purposes.

2.2.2 Integer Arithmetics

We call the first-order theory of the structure $\langle \mathbb{N}, <, +, 0, 1\rangle$ *Presburger arithmetic*. This theory was first shown to be decidable by Presburger [74] in 1929 via a quantifier elimination procedure. Note that in the case of Presburger arithmetic, we can equivalently work with the structure $\langle \mathbb{Z}, <, +, 0, 1\rangle$ since negative integers can be expressed as the difference of positive integers. Also, it might seem that we can only express inequality constraints, but we can also express equality constraints and divisibility constraints by constants in this theory. For example, the following two statements are equivalent $\exists y.x = y + y$ and $2|x$. Without any loss of generality, we can assume that Presburger formulas are presented in a *normal form* such that all atomic formulas are of the form

$$a_1x_1 + a_2x_2 + \cdots + a_kx_k = b, a_1, \ldots, a_k, b \in \mathbb{Z},$$

where $x_1, \ldots, x_k$ are arbitrary first-order variables. We write $\mathbf{0}$ to denote a tuple of zeros in any arbitrary fixed dimension and we call a language $L \subseteq (\Sigma^k)^*$ *zero-closed* if $L = \mathbf{0}^* \cdot L$. Moreover, since $<$ and $+$ are automatic relations, we can conclude that Presburger arithmetic is automatic. This is illustrated by the following lemma

Lemma 1 ([92], see also [45, Eqn. (1)])**.** *Given a system of equations $\Phi \equiv A \cdot \boldsymbol{x} = \boldsymbol{b}$ with $A \in \mathbb{Z}^{m\times d}$ and $\boldsymbol{b} \in \mathbb{Z}^m$, there is a DFA V with at most $2^m \cdot \max\{\|A\|_{1,\infty}, \|\boldsymbol{b}\|_\infty\}^m$ states such that $L(V)$ is zero-closed and $[\![L(V)]\!] = [\![\Phi]\!]$.*

Furthermore, we call the first-order theory of the structure $\langle \mathbb{N}, <, +, V_p(\cdot,\cdot), 0, 1\rangle$ *Büchi arithmetic*, where the predicate $V_p(x, y)$ holds if and only if y is the largest power of p that divides x. For convenience, we assert that $V_p(x, 0)$ never holds. We fix $p = 2$ and note that this comes with no loss of generality, as changing the base in which we encode integers does not affect the overall results. However, we recall that the first-order theory $\langle \mathbb{N}, <, +, V_p(\cdot,\cdot), V_k(\cdot,\cdot), 0, 1\rangle$, where p, k are two multiplicatively independent integers is undecidable. We note that this is also an automatic relation, so Büchi arithmetic is also automatic. We say that a predicate $R(x_1, \ldots, x_k)$ is a *regular predicate* if $R = \mathbf{0}^* \cdot L$, where $L \subseteq (\Sigma^k)^*$ is a regular language. We interpret R as $[\![R]\!] \subseteq \mathbb{N}^k$, and the additional leading zeros we require ensure that $R = [\![[\![R]\!]]\!]^{-1}$. Not only is Büchi arithmetic automatic, but it has been shown that Büchi arithmetic can express any regular predicate, i.e., a set $[\![M]\!] \in \mathbb{N}^n$ is definable in Büchi arithmetic if and only if M is a regular language, see for example [17], and [18]. We note however that in both examples, the full theory of Büchi arithmetic (with quantifier alternation) is needed to be able to express any regular predicate.

Another extension of Presburger arithmetic is the first-order theory of the structure $\langle \mathbb{N}, 0, 1, +, 2^x\rangle$ which we refer to as *Semënov arithmetic*, where 2^x is the power relation of base two, consisting of all tuples $(a, b) \in \mathbb{N}^2$ such that $b = 2^a$. It is known that Semënov arithmetic is decidable and also admits quantifier elimination [12, 21, 83]. However, this theory is not automatic. Consider the family of formulas $\Phi_n \equiv x_1 = 1 \wedge \bigwedge_{1\leq i\leq n} x_{i+1} = 2^{x_i} \wedge y < x_n$. Then the (finite) number of solutions of Φ_n is a tower of height n. Suppose Semënov arithmetic was automatic. Then each of its relations is definable by a deterministic finite automaton (DFA) of size at most m for some $m \in \mathbb{N}$ over some alphabet Σ. But then the DFA for Φ_n is acyclic and has at most m^{n+2} many states, meaning that Φ_n has at most $(m^{(n+2)}\,|\Sigma|)^{(m^{n+2})}$ different solutions, a contradiction[1]. Extending Semënov arithmetic with the Büchi $V_P(\cdot,\cdot)$ predicate breaks decidability, see e.g., [73]. However, this undecidability result requires an $\exists^*\forall^*$-quantifier prefix and does not rule out decidability of the existential fragment. Thus, we define the first-order theory of $\langle \mathbb{N}, 0, 1, +, 2^x, \{R_i\}_{i\geq 0}\rangle$, where $R_0, R_1, \ldots$ is an enumeration of all regular languages over the alphabets $\{0, 1\}^k$, $k \geq 1$, as *generalised Semënov arithmetic*. As stated, this theory is undecidable, but we prove the decidability of the existential fragment in Chapter 3.

Theorem 2. *The existential fragment of generalised Semënov arithmetic is decidable in EXPSPACE.*

[1]This elegant argument was suggested by an anonymous reviewer during the submission phase for STACS24'.

For presentational convenience, atomic formulas of generalised Semënov arithmetic are one of the following:

- linear equations of the form $a_1 \cdot x_1 + \cdots + a_d \cdot x_d = b$, $a_i, b \in \mathbb{Z}$,
- exponential equations of the form $x = 2^y$, and
- $R_i(x_1, \ldots, x_k)$, for R_i a k-ary regular predicate.

Here, $x_1, \ldots, y$ are arbitrary first-order variables. Clearly, richer atomic formulas such as $x + 2^{2^y} + y = z + 5$ can be defined from those basic class of atomic formulas, since, in this example, the formula $x + 2^{2^y} + y = z + 5$ is equivalent to the formula $\exists u \exists v\, u = 2^v \wedge v = 2^y \wedge x + u + y - z = 5$. Moreover, since we are interpreting numbers over non-negative integers, we can define the order relation in existential generalised Semënov arithmetic. This enables us without loss of generality to assume that existential formulas of Semënov arithmetic are positive, since $\neg(x = y) \equiv x < y \vee y < x$ and $\neg(x = 2^y) \equiv \exists z\, z = 2^y \wedge \neg(x = z)$. Also, consider a regular predicate $R = \mathbf{0}^* \cdot L$. Since regular languages are closed under complement, the negation $\neg R(x_1, \ldots, x_k)$ can be rewritten as $R^C(x_1, \ldots, x_k)$, where $R^C = \mathbf{0}^* L^C$.

The *size* of an atomic formula $R(x_1, \ldots, x_k)$ of generalised Semënov arithmetic, where R is a regular predicate is defined as the number of states of the canonical minimal DFA defining R. For all other atomic formulas φ, we define its size $|\varphi|$ as the number of symbols that are necessary to write it down, where we assume a binary encoding of numbers. We use binary encoding of numbers since we can "simulate" it by introducing new existentially quantified variables. The formula $x = 8$ can be expressed as $x = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1$ or as $x_0 = 1 \wedge x_1 = x_0 + x_0 \wedge x_2 = x_1 + x_1 \wedge x = x_2 + x_2$. The size $|\Phi|$ of an arbitrary existential formula Φ of generalised Semënov arithmetic is the sum of the sizes of all atomic formulas of Φ.

2.2.3 Rudimentary Arithmetic

It is a well-known fact that the first-order theory of the structure $\langle \mathbb{N}, +, \times \rangle$ is undecidable since it can encode, for example, *Hilbert's tenth problem*, which asks given polynomial $p : \mathbb{R}^n \to \mathbb{R}$ with integer coefficients, whether or not there exists an integer root. One way get around this issue is to restrict the theory by removing multiplication, e.g., Presburger arithmetic. Another way is to restrict the universe to initial segments of $\mathbb{N}$, which we call *rudimentary arithmetic*. We denote by $\mathbf{FO}(+, \times)$ the first-order theory of the structure $\langle \mathbb{N}, +, \times \rangle$, interpreted over initial segments of $\mathbb{N}$, i.e., sets $\{0, 1, \ldots, N\}$, for some fixed $N \in \mathbb{N}$. Note that, in particular, for a predicate

$x + y = z$ to hold, all of x, y, z must be at most N. It thus might seem that after we fix N, a formula $\Phi(\boldsymbol{x})$ can only express facts about numbers up to N. However, as discussed in [82] and [39], this can be improved to quantifying over variables up to N^d for any fixed d using $(N+1)$-ary representations of numbers. In other words, for any fixed d and formula $\Phi(\boldsymbol{x})$, there exists a formula $\Phi'(\boldsymbol{x})$ such that for any $N \in \mathbb{N}$ and $\boldsymbol{x} \in \underline{N}^n$, we have that $\langle \underline{N}, +, \times \rangle \models \Phi'(\boldsymbol{x})$ iff $\langle \underline{N}^d, +, \times \rangle \models \Phi(\boldsymbol{x})$. It is interesting to note that many important relations such as primality can still be expressed in this theory.

Rudimentary arithmetic can be extended with counting quantifiers. As described in [82], let rudimentary $\mathbf{FOunC}(+, \times)$ be rudimentary $\mathbf{FO}(+, \times)$ extended with counting quantifiers of the form $\exists^{=x} y \ \varphi(y)$. In this expression, the variable x is free and the variable y is bounded by the quantifier. The semantics of this expression is that there exist exactly x different values of y such that the formula $\varphi(y)$ is satisfied.

Moreover, $\mathbf{FOunC}(+, \times)$ can be extended to $\mathbf{FO}k\text{-}\mathbf{aryC}(+, \times)$, which is $\mathbf{FO}(+, \times)$ with k-ary counting quantifiers $\exists^{=\boldsymbol{x}} \boldsymbol{y} \ \varphi(\boldsymbol{y})$. In this expression, $\boldsymbol{x}, \boldsymbol{y}$ are vectors of the same dimension, and similarly to the previous case, all the variables of $\boldsymbol{x}$ are free and all the variables of $\boldsymbol{y}$ are bounded by the quantifier. The semantics is that the k-tuple $\boldsymbol{x}$ is the $(N+1)$-ary representation of the number of k-tuples $\boldsymbol{y}$ that satisfy $\varphi(\boldsymbol{y})$. As shown in [10], rudimentary $\mathbf{FOunC}(+, \times)$ and rudimentary $\mathbf{FO}k\text{-}\mathbf{aryC}(+, \times)$ have the same expressive power. In Chapter 4, we describe a reduction from the following problem

Problem 3. RUDIMENTARY $\mathbf{FO}k\text{-}\mathbf{aryC}(+, \times)$ VALIDITY
Input: $\Phi(\boldsymbol{x}) \in \mathbf{FO}k\text{-}\mathbf{aryC}(+, \times)$, $N \in \mathbb{N}$, *and* $\boldsymbol{x} \in \underline{N}^n$.
Output: *YES if and only if* $\langle \underline{N}, +, \times \rangle \models \Phi(\boldsymbol{x})$.

to the VASS reachability problem which we define in the next section.

2.3 Infinite-state Systems

In this section, we present the formal models that we study. We start by defining counter automata and then multiple extensions of VASS that can be seen as counter automata that do not have access to zero-tests. Next, we define CRAs that have some more involved update functions, so we take extra care when defining those.

2.3.1 Counter automata and VASS

Let *Ops* be the set of all integer functions of the type $f(x) = x + z$, with $z \in \mathbb{Z}$. For any integer $d \geq 0$, a *d-counter automaton* is a tuple $\mathcal{A} = \langle Q, \Delta, \lambda, \zeta, q_0, F \rangle$, where Q is a finite set of *control states*, $\Delta \subseteq Q \times Q$ is a finite set of *transitions*, $\lambda \colon \Delta \to Ops^d$ is the *update function*, $\zeta \colon \Delta \to [1, d] \cup \{\top\}$ is a function indicating which counter is tested for zero along a transition ($\top$ meaning no counter is tested), $q_0 \in Q$ is the *initial control state*, $F \subseteq Q$ is the set of *final control states*.

The set of configurations of $\mathcal{A}$ is $C(\mathcal{A}) := \{(q, n_1, \ldots, n_d) : q \in Q, n_1, \ldots, n_d \in \mathbb{N}\}$. The *initial configuration* of $\mathcal{A}$ is $c_0 = (q_0, 0, \ldots, 0)$. A d-counter automaton $\mathcal{A}$ induces a directed *configuration graph* $G = (C(\mathcal{A}), \rightarrow)$, where $\rightarrow \subseteq C(\mathcal{A}) \times C(\mathcal{A})$ such that $c \xrightarrow{t} c'$ if and only if $c = (q, n_1, \ldots, n_d)$, $c' = (q', n'_1, \ldots, n'_d)$, and $t = (q, q') \in \Delta$ such that $\lambda(t) = (f_1, \ldots, f_d)$, and $n'_i = f_i(n_i)$, for all $1 \leq i \leq d$. A run π of a counter automaton $\mathcal{A}$ from configuration $c_1 \in C(\mathcal{A})$ to $c_{n+1} \in C(\mathcal{A})$ is a sequence of configurations interleaved with transitions

$$\pi = c_1 \xrightarrow{t_1} c_2 \xrightarrow{t_2} \ldots \xrightarrow{t_n} c_{n+1}.$$

For $1 \leq i \leq j \leq n$, we denote by $\pi[i, j]$ the subsequence $c_i \xrightarrow{t_i} c_{c+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} c_j$. We denote by $\mathrm{val}(\pi, i)$ the value n_i of the i-th counter in the last configuration of π.

Observe that we can assume without loss of generality that each transition $t \in \Delta$ is either an *update transition*, meaning that no counter is tested for zero ($\zeta(t) = \top$), or a *zero-test transition*, meaning that t does not change the values of the counters ($\zeta(t) = j$, for some $1 \leq j \leq d$ and $\lambda(t) = (f, \ldots, f)$, where $f(x) = x$).

We say that $\mathcal{A}$ is a *vector addition system with states of dimension d (d-VASS)* if $\mathcal{A}$ has no zero-test transitions, i.e., ζ is the constant function assigning $\top$ to all transitions. We are now ready to formally introduce the VASS reachability problem.

Problem 4. VASS REACHABILITY
Input: *A VASS $\mathcal{V}$ and two configurations $c, c' \in C(\mathcal{V})$.*
Output: *YES if and only if $\mathcal{V}$ has a run from c to c'.*

We say that a $\mathbb{Z}$-*VASS* is a VASS for which the counters are allowed to take negative values, i.e. , for a $\mathbb{Z}$-VASS $\mathcal{V}$, the configuration space of $\mathcal{V}$ is $C(\mathcal{V}) = \{(q, z_1, \ldots, z_d) : q \in Q, z_1, \ldots, z_d \in \mathbb{Z}\}$. Furthermore, a VASS *with resets* is a VASS that can also reset the value of a counter, meaning that we extend the set *Ops* with the constant function $f(x) = 0$.

Next, we define a *labelled affine vector addition system of dimension d (d-LAVASS)*. Informally, a d-LAVASS is a labelled d-VASS, where we also allow affine updates on

the counters. Formally, for $d \geq 0$, a d-LAVASS is a tuple $\mathcal{V} = \langle Q, \Sigma, \Delta, \lambda, q_0, F, \Phi \rangle$, where Q is a finite set of control states, Σ is a *finite alphabet*, $\Delta \subseteq Q \times \mathcal{P}(\Sigma) \times Q$ is a finite set of transitions, $\lambda : \Delta \to Ops^d$ is the update function, where Ops is now defined as the set of all affine functions of type $f(x) = ax + b$, with $a, b \in \mathbb{Z}$, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of finite control states, and Φ is a a quantifier-free formula of Presburger arithmetic $\Phi(x_1, \ldots, x_d)$ that specifies an infinite set $[\![\Phi]\!] \subseteq \mathbb{N}^d$ of *final counter values*. Note that when $d = 0$, then, $\mathcal{V}$ is essentially a non-deterministic automaton.

As before, a LAVASS $\mathcal{V}$ induces an (infinite) labelled directed *configuration graph* $G = (C(\mathcal{V}), \to)$, where $\to \subseteq C(\mathcal{V}) \times \Sigma \times C(\mathcal{V})$ such that $c \xrightarrow{a} c'$ if and only if $c = (q, m_1, \ldots, m_d)$, $c' = (q', m'_1, \ldots, m'_d)$, and there is $t = (q, a, q') \in \Delta$ such that $a \in \Sigma$, $\lambda(t) = (f_1, \ldots, f_d)$, and $m'_i = f_i(m_i)$ for all $1 \leq i \leq d$. We lift the definition of $\to$ to words $w = a_1 \cdots a_n \in \Sigma^*$ in the natural way, and thus write $c \xrightarrow{w} c'$ whenever $c \xrightarrow{a_1} c_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} c_{n-1} \xrightarrow{a_n} c'$ for some $c_1, \ldots, c_{n-1} \in C$. The *language* $L(\mathcal{V}) \subseteq \Sigma^*$ of $\mathcal{V}$ is defined as

$$L(\mathcal{V}) := \{w \in \Sigma^* : c_0 \xrightarrow{w} c_f, c_f \in C_f(\mathcal{V})\}.$$

The *non-emptiness problem* for a LAVASS is the following

Problem 5. LAVASS NON-EMPTINESS
Input: *A LAVASS $\mathcal{V}$.*
Output: *YES if and only if $L(\mathcal{V}) \neq \emptyset$.*

2.3.2 Cost Register Automata

VASS are infinite-state systems in which each counter is updated based on its previous value without having access to the values of the other counters. Informally, *Cost Register Automata (CRA)* are labelled infinite-state systems that do not have this restriction. Note that in the case of CRAs we refer to counters as registers. This is mainly a convention, but it also helps to differentiate between counters that do not interact with each other and registers which do interact with each other.

Since the update functions in the case of CRAs are more complex than in the case of affine updates, we take extra care when defining them. Formally, the update functions of a CRA are *expressions* over a semiring $\mathbb{K}$. In this thesis, we fix $\mathbb{K} := (\mathbb{Z} \cup \{+\infty\}, \min, +)$ to be the tropical semiring. Also, we fix X to be a finite set of variables, and $d := |X|$. By $\mathrm{Expr}(X)$ we denote the set of expressions constructed using operations $\min\{\cdot, \cdot\}$ and $+$, variables from X, and constants from $\mathbb{K}$. Due to

associativity, we allow arbitrary arity of the semiring operations in the expression notation. A *substitution over* X is a function $\nu\colon X \to \mathrm{Expr}(X)$. We denote the set of all substitutions over X by $\mathrm{Sub}(X)$. When defining substitutions, we often treat them as sets of '*argument* $\leftarrow$ *value*' pairs. When X is clear from the context, we implicitly extend partial substitutions with the identity mapping for the omitted arguments. For example, if $X' = \{x_1, x_2\} \subsetneq X$, then $\nu = \{x_1 \leftarrow e_1,\ x_2 \leftarrow e_2\}$ denotes a substitution satisfying $\nu(x_i) = e_i$ for $x_i \in X'$ and $\nu(x) = x$ for $x \in X \setminus X'$. A *valuation over* X is a substitution $\mu\colon X \to \mathbb{K}$. We denote by $\mathrm{Val}(X) \subset \mathrm{Sub}(X)$ the set of all valuations over X. We abuse notation slightly and write $\mathbf{0}$ for the valuation that assigns the value 0 to all $x \in X$.

Example 6. *Fix* $X := \{x, y, z\}$.

$$
\begin{aligned}
e &:= \min\{10, x+5, y+z\} && \in \mathrm{Expr}X && \text{(an expression)}\\
\nu &:= \{x \leftarrow 2 + \min\{x, y\},\ y \leftarrow 3\} && \in \mathrm{Sub}(X) && \text{(a substitution)}\\
\mu &:= \{x \leftarrow 2,\ y \leftarrow 5,\ z \leftarrow 10\} && \in \mathrm{Val}X && \text{(a valuation)}
\end{aligned}
$$

Substitutions ν, ν' can be *composed*, $\nu\nu' \in \mathrm{Sub}(X)$, by simultaneously replacing each occurrence of x with $\nu(x)$, in each expression of ν', for every $x \in X$. Thus, composing a valuation μ with a substitutions ν results in another valuation, so we write $\mathrm{eval}_\mu(\nu)$ to emphasis this and note that in this case, we can equivalently consider $\mathrm{eval}_\mu(\nu) \in \mathbb{K}^d$ as a vector. Additionally, for $\nu, \nu' \in \mathrm{Sub}(X)$, we write $\nu \equiv \nu'$ whenever $\nu(x) \equiv \nu'(x)$ for every $x \in X$. For an expression e, by $\mathrm{maxc}(e)$ we denote the maximal absolute value of constants different to $+\infty$ appearing in e, and 0 if there are none. We extend maxc naturally to substitutions.

Similarly to the case of VASS, we restrict the set of substitutions that we study in this thesis. We focus on a special family of substitutions which are *copyless* and *linear with resets*. A substitution ν is *linear with resets* if for every $x \in X$, $\nu(x)$ is either a linear combination (over the tropical semiring) of X or 0 (a *reset*). A linear substitution with resets ν is *copyless* if for all pairs $x, x' \in X$, such that $x \neq x'$, the expressions $\nu(x)$ and $\nu(x')$ feature disjoint sets of variables. By $\mathrm{Sub}_{\mathrm{clr}}$ we denote the set of all copyless linear substitutions.

Example 7 (Copyless linear substitutions with resets)**.** *Consider the substitutions* ν *and* ν'*:*

$$
\nu := \begin{cases} x \leftarrow y + 5 \\ y \leftarrow \min\{x, z-2\} \\ z \leftarrow 5 \end{cases}
\qquad
\nu' := \begin{cases} x \leftarrow \min\{x\} \\ y \leftarrow \min\{x, y\} \end{cases}
$$

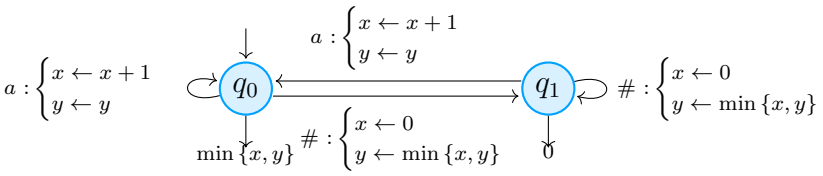


Figure 2.1: An example of a CRA over $\mathbb{K}$

We have that $\nu \in \mathrm{Sub}_{\mathrm{clr}}(\{x, y, z\})$ *because all updates are linear and variables* x, y, z *occur at most once in its expressions. In contrast,* ν' *is linear, but not copyless, because* x *occurs in both* $\nu'(x)$ *and* $\nu'(y)$*.*

A *copyless linear CRA with resets over* $\mathbb{K}$ is a tuple $\mathcal{C} = \langle X, \Sigma, Q, q_{\mathrm{ini}}, \Delta, \mathrm{out}\rangle$ consisting of a finite set X of registers, a finite alphabet Σ, a finite set Q of control states, an initial state $q_{\mathrm{ini}} \in Q$, and two functions:

- a transition rule function $\Delta\colon Q \times \Sigma \to \mathrm{Sub}_{\mathrm{clr}}(X) \times Q$,
- an output function $\mathrm{out}\colon Q \to \mathrm{Expr}(X)$.

For our purposes, we can assume that $+\infty$ never occurs as a constant in a substitution, since every such transition can either be removed from the minimum, or be replaced with a transition leading to a new state, a self-loop incrementing the value of the corresponding register in this new state, and then a transition back. In this thesis, we consider only copyless linear CRAs with resets, hence for the rest of the paper we simply call them CRAs.

Similarly to LAVASS, the set of configurations of $\mathcal{C}$ is $C(\mathcal{C}) := Q \times \mathrm{Val}(X)$. The initial configuration of $\mathcal{C}$ is $c_{\mathrm{ini}} := (q_{\mathrm{ini}}, \mathbf{0})$. A CRA $\mathcal{C}$ induces an (infinite) labelled directed *configuration graph* $G = (C(\mathcal{C}), \to)$, where $\to \subseteq C(\mathcal{C}) \times \Sigma \times \mathrm{Sub}_{\mathrm{clr}} \times C(\mathcal{C})$ such that $c \xrightarrow{a,\nu} c'$ if and only if $c = (q, \mu)$, $c' = (q', \mu')$, and there is $a \in \Sigma$ such that $\Delta(q, a) = (\nu, q')$ and $\mathrm{eval}_\mu(\nu) = \mu'$. We lift the definition of $\to$ to words $w = a_1 \cdots a_n \in \Sigma^*$ in the natural way. We further define the function $\mathcal{C} : \Sigma^* \to \mathbb{K}$ computed by $\mathcal{C}$ as $\mathcal{C}(w) := \mathrm{eval}_\mu(\mathrm{out}(q))$, such that $c_{ini} \xrightarrow{w} (q, \mu)$.

Example 8. *In Figure 2.1, we give an example of a CRA over the semiring* $\mathbb{K}$ *and alphabet* $\Sigma = \{a, \#\}$*. It has two states* q_0 *and* q_1*, where* q_0 *is the initial state, and two registers* x *and* y*. The output function for state* q_0 *(words ending with* a *and the empty word) is* $\min\{x, y\}$ *and for state* q_1 *(words ending with* $\#$*) is* 0*. Both registers are initialised with value* 0*. If the input word ends with* $\#$ *or is empty, this CRA outputs* 0*. If it ends with* a*, the CRA outputs the length of the shortest maximal by*

inclusion blocks of consecutive a's. Register x stores the number of a's read in the current block, and y stores the minimum length of the blocks read so far. This CRA is a copyless linear CRA with resets.

Finally, we say that a CRA $\mathcal{C}$ is *bounded* if there is $N \in \mathbb{N}$ such that $\mathcal{C}(w) < N$, for all $w \in \Sigma^*$. The problem that we study in Chapter 5 is the following

Problem 9. CRA BOUNDEDNESS
Input: *A CRA $\mathcal{C}$.*
Output: *Yes if and only if C is bounded.*

Chapter 3

Semënov Arithmetic and LAVASS

In this chapter, we study the complexity of deciding the existential fragment of generalised Semënov arithmetic. We establish that this problem can be solved in EXPSPACE, Theorem 2, by finding a correspondence between existential formulas of generalised Semënov arithmetic and a restricted class of LAVASS. Thus, we are able to find a reduction to the language emptiness problem of this restricted class.

We start this chapter by proving several closure properties of the languages of LAVASS and explaining how a LAVASS can, in some sense, model an existential formula of generalised Semënov arithmetic. Next, we observe that we do not need the full power of LAVASS in order to accomplish this modelling, so we define a restricted class of LAVASS that is sufficient for this purpose and for which we can decide the language emptiness problem. Not only is the emptiness language problem for LAVASS undecidable, which we show in this chapter, but we are mainly interested in deciding existential generalised Semënov.

Then, the main part of the chapter is devoted to proving that the emptiness problem for this restricted class can be decided in EXPSPACE. The proof is quite technical and makes use of some intricate path schemes. Thus, before presenting the full proof, we provide an overview to give the reader the main ideas used inside the proof.

3.1 Related Work

One of the main research directions in the field of arithmetic theories is searching for more expressive extensions of well-studied theories while maintaining decidability. This is especially true in the case of Presburger arithmetic. The task however is not trivial since, for example, adding the multiplication function to the theory immediately results in undecidability. Moreover, adding other predicates that might

seam weaker than full multiplication, such as the squaring function, leads to undecidability since in this case, the multiplication function can be recovered inside the theory. For more examples of predicates that lead to undecidability, and also some decidable expansions of Presburger arithmetic see, for instance, the survey [14]. Famously, Hilbert's 10th problem asks if there exists an algorithm that can decide if a polynomial equation with integer coefficients has an integer root. This problem was shown to be undecidable by Matiyasevich in 1970, in [64]. Since the problem can be encoded in the existential fragment of the structure $\langle \mathbb{Z}, 0, 1, +, \times, < \rangle$, this implies that the existential fragment of the extension of Presburger arithmetic with multiplication is also undecidable.

The decidability of an arithmetic is usually proved in one of two ways. Either via a quantifier elimination procedure as in the case of the original proof by Presburger from 1929 [74], or via the notion of automaticity. An important example of the latter is the proof by Büchi from 1960, who showed that the extension of Presburger with the $V_2(\cdot)$ predicate is decidable [19]. It was however shown that, for multiplicatively independent integers k, l, the first-order theory of the structure $\langle \mathbb{N}, 0, 1, +, <, V_k, V_l \rangle$ is undecidable [88]. After Büchi, Semënov used the method of quantifier elimination to prove that any extension of Presburger arithmetic with an "effectively sparse" predicate is decidable [83]. One important example of a sparse predicate is $2^{\mathbb{N}}$ which is the set of all integers that are powers of two. Although the predicate $2^{\mathbb{N}}$ can be defined inside $\langle \mathbb{N}, 0, 1, +, <, V_2 \rangle$, the quantifier elimination procedure given by Semënov applies more broadly. More recently, it was shown that extending Presburger with two power predicates leads to undecidability. Namely, it was shown in [49], that given two integers $k, l \geq 2$, the first-order theory of $\langle \mathbb{N}, 0, 1, +, <, k^{\mathbb{N}}, l^{\mathbb{N}} \rangle$ is undecidable. However, it is proved in [52] that the existential fragment of this theory is decidable.

There was also some focus on the computational complexity of these decision procedures. The first improvement of Presburger's procedure was done by Cooper in 1972 [23] which enabled it to be applied to non-trivial examples. Then it was shown that Cooper's algorithm runs in triply exponential time [72]. A finer-grained complexity analysis for the complexity of Presburger arithmetic is presented in [90] for example. There is a recent result studying the complexity of Presburger arithmetic extended with the power predicate or with the exponentiation function. Concretely, it was shown in [12] that the complexity of deciding $\langle \mathbb{N}, 0, 1, +, <, 2^{\mathbb{N}} \rangle$ is in 3EXPTIME, and the complexity of deciding the existential fragment of $\langle \mathbb{N}, 0, 1, +, <, 2^x \rangle$ is in NEXPTIME.

3.2 Affine vector addition systems with states

In this section, we discuss closure properties of LAVASS and show that their languages are closed under union and intersection, and restricted kinds of homomorphisms and inverse homomorphisms, using essentially the standard constructions known for finite-state automata. Before we proceed, we need to start by defining the size of a LAVASS. Let $\mathcal{V} = \langle Q, \Sigma, \Delta, \lambda, q_0, F, \Phi \rangle$ be a LAVASS. For an update function $\lambda \colon \Delta \to Ops^d$, we define

$$\|\lambda\| := \max\{|a| + |b| : \lambda(t) = (f_1, \ldots, f_d), f_i = ax + b, 1 \leq i \leq d, t \in \Delta\}\,.$$

We define the size $|\mathcal{V}|$ of a LAVASS $\mathcal{V} = \langle Q, d, \Sigma, \Delta, \lambda, q_0, F, \Phi \rangle$ as

$$|\mathcal{V}| := |Q| + |\Delta| \cdot (d+1) \cdot \log(\|\lambda\| + 1) + |\Phi|\ .$$

Now, let $\mathcal{V}_i = \langle Q_i, d_i, \Sigma, \Delta_i, \lambda_i, q_0^{(i)}, F_i, \Phi_i \rangle$, $i \in \{1, 2\}$, be two LAVASS.

Proposition 10. *The languages of LAVASS are closed under union and intersection. Moreover, for $\mathcal{V}$ such that $L(\mathcal{V}) = L(\mathcal{V}_1) \cap L(\mathcal{V}_2)$, we have $|\mathcal{V}| \leq |\mathcal{V}_1| \cdot |\mathcal{V}_2|$.*

Closure under union is trivial since we allow for non-determinism. The result for intersection can be obtained by generalising the standard constructions known from non-deterministic finite-state automata. The set of control states of the LAVASS $\mathcal{V}$ accepting the intersection of LAVASS $\mathcal{V}_1$ and $\mathcal{V}_2$ is $Q_1 \times Q_2$. The dimension of $\mathcal{V}$ is the sum of the dimensions of $\mathcal{V}_1$ and $\mathcal{V}_2$, and the counters of $\mathcal{V}_1$ and $\mathcal{V}_2$ get independently simulated in the counters of $\mathcal{V}$. Upon reading an alphabet symbol a, the LAVASS $\mathcal{V}$ then simultaneously simulates the respective transitions of $\mathcal{V}_1$ and $\mathcal{V}_2$ for a. More formally, to show closure under intersection, we define the LAVASS $\mathcal{V} := (Q', d_1 + d_2, \Sigma, \Delta', \lambda', q_0', F', \Phi)$ such that

- $Q' := Q_1 \times Q_2$,
- $((q_1, q_2), a, (r_1, r_2)) \in \Delta'$ if and only if $(q_1, a, r_1) \in \Delta_1$ and $(q_2, a, r_2) \in \Delta_2$,
- $\lambda'((q_1, q_2), a, (r_1, r_2)) := (\lambda_1(q_1, a, r_1), \lambda_2(q_2, a, r_2))$,
- $q_0'' := (q_0^{(1)}, q_0^{(2)})$,
- $F' := F_1 \times F_2$, and
- Φ is the conjunction of Φ_1 and Φ_2, with counters renamed accordingly.

Lemma 11. *For any $w \in \Sigma^*$, $q^1, r^{(1)} \in Q_1$ and $q^2, r^2 \in Q_2$, the following are equivalent:*

(i) $((q^{(1)}, q^{(2)}), m_1, \ldots, m_{d_1+d_2}) \xrightarrow{w}_{\mathcal{V}} ((r^{(1)}, r^{(2)}), m'_1, \ldots, m'_{d_1+d_2})$

(ii) $(q^{(1)}, m_1, \ldots, m_{d_1}) \xrightarrow{w}_{\mathcal{V}_1} (r^{(1)}, m'_1, \ldots, m'_{d_1})$ *and* $(q^{(2)}, m_{d_1+1}, m_{d_1+1} \ldots, m_{d_1+d_2}) \xrightarrow{w}_{\mathcal{V}_2} (r^{(1)}, m'_{d_1+1}, \ldots, m'_{d_1+d_2})$.

Proof. Let $w \in \Sigma^*$, we prove the statement by induction on $|w|$. The base case $w = \epsilon$ is immediate by the definition of $\mathcal{V}$.

For the induction step, let $w = u \cdot a$ for some $a \in \Sigma$. The induction hypothesis yields

$$\begin{aligned} & ((q^{(1)}, q^{(2)}), m_1, \ldots, m_{d_1+d+2}) \xrightarrow{u}_{\mathcal{V}} ((s^{(1)}, s^2), m''_1, \ldots, m''_{d_1+d_2}) \\ \iff & (q^{(1)}, m_1, \ldots, m_{d_1}) \xrightarrow{u}_{\mathcal{V}_1} (s^{(1)}, m''_1, \ldots, m''_{d_1}) \text{ and} \\ & (q^{(2)}, m_{d_1+1}, \ldots, m_{d_1+d_2}) \xrightarrow{u}_{\mathcal{V}_2} (s^{(2)}, m''_{d_1+1}, \ldots, m''_{d_1+d_2}) . \end{aligned}$$

Again, by definition of $\mathcal{V}$ we furthermore have

$$\begin{aligned} & ((s^{(1)}, s^{(2)}), m''_1, \ldots, m''_{d_1+d+2}) \xrightarrow{a}_{\mathcal{V}} ((r^{(1)}, r^2), m'_1, \ldots, m''_{d_1+d_2}) \\ \iff & (s^{(1)}, m_1, \ldots, m_{d_1}) \xrightarrow{a}_{\mathcal{V}_1} (r^{(1)}, m''_1, \ldots, m''_{d_1}) \text{ and} \\ & (s^{(2)}, m_{d_1+1}, \ldots, m_{d_1+d_2}) \xrightarrow{a}_{\mathcal{V}_2} (r^{(2)}, m''_{d_1+1}, \ldots, m''_{d_1+d_2}) . \end{aligned}$$

This concludes the proof of the statement. □

Corollary 12. *Let $\mathcal{V}_1, \mathcal{V}_2$ be LAVASS. Then $L(\mathcal{V}_1) \cap L(\mathcal{V}_2)) = L(\mathcal{V}_1 \cap \mathcal{V}_2)$.*

Proof. For any $w \in \Sigma^*$, by Lemma 11, we have:

$$\begin{aligned} & w \in L(\mathcal{V}_1 \cap \mathcal{V}_2) \\ \iff & (q'_0, 0, \ldots, 0) \xrightarrow{w}_{\mathcal{V}} ((r_1, r_2), m_1, \ldots, m_{d_1+d_2}) \\ & \text{for some } (r_1, r_2) \in F', (m_1, \ldots, m_{d_1+d_2}) \in [\![\Phi]\!] \\ \iff & (q_0^{(1)}, 0, \ldots, 0) \xrightarrow{w}_{\mathcal{V}_1} (r_1, m_1, \ldots, m_{d_1}) \text{ for some } r_1 \in F_1, (m_1, \ldots, m_{d_1}) \in [\![\Phi_1]\!], \text{ and} \\ & (q_0^{(2)}, 0, \ldots, 0) \xrightarrow{w}_{\mathcal{V}_2} (r_2, m_{d_1+1}, \ldots, m_{d_1+d_2}) \\ & \text{for some } r_2 \in F_2 \, (m_{d_1+1}, \ldots, m_{d_1+d_2}) \in [\![\Phi_2]\!] \\ \iff & w \in L(\mathcal{V}_1) \text{ and } w \in L(\mathcal{V}_2) . \end{aligned}$$

□

From the construction, it is clear that for the size of the LAVASS for $L(\mathcal{V}_1) \cap L(\mathcal{V}_2)$, we have:

- $|Q| = |Q_1| \cdot |Q_2|$,
- $|\Delta| \leq |\Delta_1| \cdot |\Delta_2|$,
- $|\lambda| = max(|\lambda_1|, |\lambda_2|)$, and
- $d = d_1 + d_2$.

Note that since LAVASS languages contain regular languages, Proposition 10 in particular enables us to intersect LAVASS languages with regular languages.

Let Σ, Γ be two finite alphabets. Recall that a homomorphism $h\colon \Gamma^* \to \Sigma^*$ is fully defined by specifying $h(a)$ for all $a \in \Gamma$. We call h a *projection* if $|h(a)| = 1$ for all $a \in \Gamma$.

Proposition 13. *The languages of LAVASS are closed under projections and inverses of projections.*

Proof. Let $h\colon \Gamma^* \to \Sigma^*$ be a projection. Given an LAVASS $\mathcal{V} = \langle Q, d, \Sigma, \Delta, \lambda, q_0, F, S_f \rangle$, to obtain closure under projections replace any $t = (q, a, q') \in \Delta$ with $t' = (q, h(a), q')$, and set $\lambda(t') := \lambda(t)$. To obtain closure under inverse projections, replace any $t = (q, a, q') \in \Delta$ with $t' = (q, h^{-1}(a), q')$ and set $\lambda(t') := \lambda(t)$. □

3.3 Generalised Sëmenov arithmetic and LAVASS

Let $\Sigma = \{0, 1\}$. In this section, we show how given a quantifier-free formula $\Phi(x_1, \ldots, x_d)$ of Semënov arithmetic, we can construct a LAVASS $\mathcal{V}$ over the alphabet $\Sigma_d := \{0, 1\}^d$ such that $[\![L(\mathcal{V})]\!] = \{\boldsymbol{x} \in \mathbb{N}^d : \Phi(\boldsymbol{x})\}$. We will subsequently observe that the resulting LAVASS enjoy strong structural restrictions, giving rise to the fragment of restricted LAVASS that we then formally define. For our purposes, it will be sufficient to primarily focus on formulas Φ of Semënov arithmetic which are conjunctions of atomic formulas.

In Chapter 2, we stated that for presentational convenience, the atomic formulas of generalised Semënov arithmetic are either linear equations, exponential equations of the type $x = 2^y$, or regular predicates. Moreover, recall Lemma 1 which stated that the sets of solutions of systems of linear equations $A\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{Z}$ can be represented by a regular language and are hence definable via a LAVASS.

The crucial part, which requires the power of LAVASS, are exponential equations $x = 2^y$. A LAVASS $\mathcal{V}$ with two counters and $[\![L(\mathcal{V})]\!] = [\![x = 2^y]\!]$ is depicted in Figure 3.1. Control-states are depicted as circles and transitions as arrows between

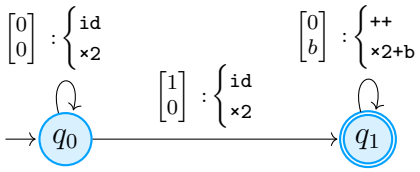


Figure 3.1: A gadget with two counters for the exponential equation $x = 2^y$, where $b \in \{0, 1\}$.

them. The vector before the colon is the alphabet symbol read. For instance, the transition from q_0 to q_1 reads the alphabet symbol $(1, 0) \in \{0, 1\}^2$. After a colon, we display the counter operations when reading the alphabet symbol, the operation on the first counter is displayed on the top and the operation on the second counter on the bottom. Here and subsequently, for presentational convenience, `id` is the identity function $x \mapsto x$, and `×2` and `×2+1` are the functions $x \mapsto 2x$ and $x \mapsto 2x + 1$, respectively.

The idea behind the gadget in Figure 3.1 is as follows. For an example, suppose that $y = 5$, then $x = 32$, and in binary the sequence of digits of x and y looks as follows:

$$\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]\left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]\left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]\cdots\left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$$

Since $x = 2^y$, we have that $x \in 0^*10^*$, and the number of trailing zeros of x is equal to the value of y. Thus, once a 1 in the binary representation of x has been read, the first counter in the gadget of Figure 3.1 keeps incrementing and counts the number of trailing zeros of x. At the same time, the second counter in the gadget of Figure 3.1 keeps the value 0 until it reads the first 1 of the binary expansion of y, since $2 \cdot 0 = 0$. It then computes the value of y in binary on the second counter by multiplying the value of the second counter by 2 when reading a zero, and multiplying by two and adding one when reading a one. The LAVASS in Figure 3.1 only accepts when the first and the second counter have the same value, i.e., when the number of trailing zeros of the binary expansion of x equals the value of y, as required.

A closer look at the gadget constructed in Figure 3.1 reveals a number of important structural properties:

(i) all operations performed on the first counter are either the identity map `id` or increments `++`;

(ii) all operations performed on the second counter are affine updates `×2` and `×2+1`;

(iii) once the first counter gets incremented on a run, it gets incremented at every subsequent transition; and

(iv) only counter configurations in which the value of the first counter equals the value of the second counter are accepted.

Those properties are crucial to obtain decidability of (generalised) existential Semënov arithmetic.

Definition 14. *An LAVASS is* restricted *if and only if it has an even number of* $2d$ *counters called* x_i, y_i*,* $1 \leq i \leq d$*, such that every counter pair* (x_i, y_i) *adheres to the above Properties (i)–(iv), where* x_i *is regarded as the first counter and* y_i *as the second counter and the set of final counter values is defined by* $\Phi \equiv \bigwedge_{1 \leq i \leq d} x_i = y_i$*.*

For convenience, when referring to the counters in a pair, we subsequently refer to the first counter as its *x-counter* and to the second counter as its *y-counter*. We also call the x-counters *unary counters* and the y-counters *binary counters*, because of the different type of updates they support. We will usually write m for the value of the x-counter and n for the value of the y-counter. The following lemma is an immediate result of Figure 3.1 together with the previous discussion.

Lemma 15. *There is a fixed restricted LAVASS* $\mathcal{V}$ *of dimension two such that* $L(\mathcal{V})$ *is zero closed and* $[\![L(\mathcal{V})]\!] = [\![x = 2^y]\!]$*.*

A discussion on these restrictions is in order. First, notice that the restrictions capture almost all of the structure of our example in Figure 3.1. This is a nice feature since we are trying to design a model that can recognise solution sets of generalised Sëmenov first-order formulas, for which the complexity of the emptiness problem is as low as possible. However, one might ask if all of these restrictions are necessary in order to obtain decidability for the emptiness problem. We argue that having affine counters and the ability to compare them in our acceptance condition is too strong of a feature. For this, we slightly modify an argument presented in [75]. We start by introducing the *Post Correspondence Problem, PCP*. Given a set of dominoes D such that each domino is a pair of binary strings (u_i, v_i), is there a way to concatenate the dominoes in order to get two matching strings? Formally, is there a sequence of indices $(i_k)_{1 \leq k \leq N}$, such that $u_{i_1} \dots u_{i_N} = v_{i_1} \dots v_{i_N}$? This problem is known to be undecidable. One can reduce the the PCP problem to the language emptiness problem for LAVASS by having two counters that model the two strings $u_{i_1} \dots u_{i_N}$ and $v_{i_1} \dots v_{i_N}$. There is a stateless LAVASS that has one transition for each domino

pair, that modifies the two counters accordingly. Next, our acceptance condition checks if these two counters equal. Of course, the counters are equal initially, but this can be easily solved by adding an extra state or requiring the counters to be greater than zero.

Thus, even a counter that has access to only `id`, `×2`, and `×2+1` updates is able to model the string obtained by concatenating domino pieces[1]. We conclude that the second restriction (ii) is mandatory if we want to be able to decide language emptiness. Next, we need a counter that is able to "count", so we would like to have access to `id` and `++` updates and we established that we cannot have affine updates together with the `id` update, hence the first restriction. The third restriction (iii) is more interesting. While it does make the proof easier which will be observed in the coming sections, it is unknown if it actually decreases the complexity of the emptiness problem. We expand on this point in the last section. Finally, the last restriction (iv) is there mainly for presentation purposes. We can easily compare two binary counters with each other by using two extra unary counters, so it is easy to see that the complexity of the language emptiness problem does not change if we allow Θ to be a conjunctions of literals of the type $c_i \leq c_j + b$, where c_i, c_j are arbitrary counters and $b \in \mathbb{Z}$. Now, we prove that the same closure properties of LAVASS carry over to the restricted class.

By following the constructions in Section 3.2 and noting that the counters of different LAVASS are simulated independently we obtain the following result.

Proposition 16. *The languages of restricted LAVASS are closed under union, intersection, projection, and inverse projections.*

Thus, we are ready to present the central lemma of this section.

Lemma 17. *Consider a positive conjunctive formula of Semënov arithmetic*

$$\Phi(\boldsymbol{x}) \equiv A \cdot \boldsymbol{x} = \boldsymbol{b} \wedge \bigwedge_{i \in I} x_i = 2^{y_i},$$

where $A \in \mathbb{Z}^{m \times n}$, $\boldsymbol{b} \in \mathbb{Z}^m$, I is a finite index set, and x_i and y_i are variables from $\boldsymbol{x}$. There is a restricted LAVASS $\mathcal{V}$ of dimension $2\,|I|$ and of size $(\|A\|_{1,\infty} + \|\boldsymbol{b}\|_\infty + 2)^{O(m+|I|)}$ such that $[\![L(\mathcal{V})]\!] = [\![\Phi]\!]$.

[1] This problem was asked by the author, and then answered by Petra Wolf at Autoboz22' (https://automata.exchange/22.02-paired-counter-automata/), before discovering that it was already treated in [75].

This lemma follows from the combination of Lemma 1, Lemma 15, and Proposition 16.

Finally, for technical convenience, we assume that for a restricted LAVASS, we have $|Q| \geq 2$. This is with no loss of generality, since if $|Q| = 1$ then deciding non-emptiness is trivial (the restricted LAVASS has non-empty language if and only if the only control state is accepting).

The next sections will be devoted to the proof of the main result of this chapter on restricted LAVASS.

Proposition 18. *Language emptiness of a restricted LAVASS $\mathcal{V}$ with $2d$ counters is decidable in* NSPACE$(|\mathcal{V}| \cdot 2^{O(d)})$.

This proposition enables us to prove Theorem 2, by appealing to Lemma 17. Given a formula Φ of generalised Semënov arithmetic, we can in space $2^{O(|\Phi|)}$ construct the disjunctive normal form of Φ. Every disjunct can be assumed to be of the form

$$A \cdot \boldsymbol{x} = \boldsymbol{b} \wedge \bigwedge_{i \in I} x_i = 2^{y_i} \wedge \bigwedge_{j \in J} R_j(\boldsymbol{x}),$$

where the R_j are predicates over regular languages. By Lemma 17, there is a restricted LAVASS for Φ of dimension $2\,|I|$ with a number of states bounded by $(\|A\|_{1,\infty} + \|\boldsymbol{b}\|_\infty + 2)^{O(m+|I|)} = 2^{p(|\Phi|)}$ for some polynomial p and whose language represents the set of solutions to $A \cdot \boldsymbol{x} = \boldsymbol{b} \wedge \bigwedge_{i \in I} x_i = 2^{y_i}$. Intersecting with the DFA for the R_j results in a restricted LAVASS $\mathcal{V}$ with $2\,|I| = O(|\Phi|)$ counters such that $|\mathcal{V}| = 2^{p(|\Phi|)}$ for some polynomial p. By Proposition 18, it follows that emptiness of $\mathcal{V}$ is decidable in NSPACE$(2^{p(|\Phi|)} \cdot 2^{2|I|})$. We conclude the argument by recalling that NEXPSPACE=EXPSPACE by Savitch's theorem. This proves Theorem 2, that states that existential generalised Semënov can be decided in EXPSPACE.

3.4 Certificates witnessing non-emptiness of restricted LAVASS

We now show that language emptiness for restricted LAVASS is decidable in exponential space, Proposition 18. Clearly, this problem reduces to deciding whether a given restricted LAVASS has an accepting run, but witnessing runs may be of non-elementary length. Below we present such an example.

Assume that all the unspecified transitions contain a `×2` update for the y-counters. Thus, the counter y_n has value 1 when we reach the state q_n, and value 3 when we

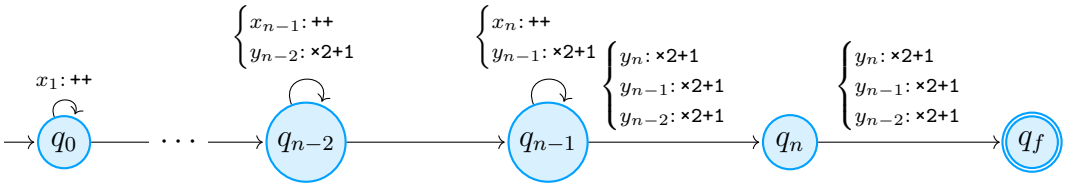


Figure 3.2: Accepting run of non-elementary length

reach the accepting state q_f. Any accepting run must also ensure that the counter x_n has value 3 when we reach q_f, thus the run has to take the self-loop in state q_{n-1} at least once. However, this increases the value of y_{n-1} which in turn forces any accepting run to use the self-loop in state q_{n-2}. Observe that, if a run uses the self-loop at state q_i at least l_i many times, it must then use the self-loop at state q_{i-1} more than 2^{l_i} many times in order to recover the difference between the counters y_i and x_i. Thus, the shortest accepting run must be of non-elementary length.

To overcome this problem, we define an abstraction for configurations of restricted LAVASS. Abstract configurations store residue classes of counter values, as well as some further information that is required to witnesses the existence of concrete accepting runs. Before giving the formal definition, we provide some high level intuition that leads to our definition of abstract configurations. Next, we introduce reachability certificates, which are abstract runs with certain further properties. We argue that the existence of *witnessing certificates*, which are special kinds of reachability certificates witnessing that the language of an LAVASS is non-empty, are decidable in EXPSPACE. The last subsection then establishes that witnessing certificates actually witness non-emptiness of restricted LAVASS.

3.4.1 Key observations of accepting runs

Given a *restricted* LAVASS $\mathcal{V}$ in dimension d, assuming that $L(\mathcal{V}) \neq \emptyset$, there is a run π from an initial configuration c to a final configuration c'. With no loss of generality, throughout this section, we assume that $\mathrm{val}(c', x_i) \geq \mathrm{val}(c', x_{i+1}) > 0$ for all $1 \leq i < d$. Recall that the notation refers to the unary counters, the ones that keep track of the number of zeros read after the first 1. In particular, this implies that every counter gets incremented at least once along a path witnessing non-emptiness.

Our first observation is that if, along π, a counter y_i achieves the first time a non-zero value by taking a `×2+1` labeled transition, then the length of the remaining segment of π is bounded by $O(\log(m_i + 1))$, where m_i is the value of counter x_i before the transition is taken. The reason is that, once y_i has non-zero value, its value is at

least doubling for every following transition taken. Hence if π is "long" then along π there will be loops incrementing a counter x_i before the corresponding y_i achieves non-zero value.

In the latter scenario, we may actually, subject to some bookkeeping, discard concrete values of x_i and y_i and only store their residue classes modulo ℓ_i, where ℓ_i is the length of the first loop incrementing x_i along π. In particular, if we are given a non-accepting run π' such that $\mathrm{val}(\pi', x_i) \equiv \mathrm{val}(\pi', y_i) \bmod \ell_i$ and $\mathrm{val}(\pi', x_i) < \mathrm{val}(\pi', y_i)$ then π' can be turned into a run π'' where $\mathrm{val}(\pi'', x_i) = \mathrm{val}(\pi'', y_i)$ by iterating the loop of length ℓ_i.

There are, however, some further subtleties that need to be taken care of. Consider the segment π' of π between the first transition labelled by `++` on x_i and the first transition labelled by `++` on x_{i+1}. If π' contains no loop, then, we are in a situation where the first loop incrementing x_i is also the first loop incrementing x_{i+1}. This means that the values of x_i and x_{i+1} get paired together, and hence, for an accepting run, also the values of y_i and y_{i+1} are paired together. In our approach, we deal with such circumstances by introducing so-called *y-constraints*. A y-constraint of the form $y_i - y_{i+1} = \delta_i$ for some constant $\delta_i \in \mathbb{N}$ asserts that the counters y_{i+1} and y_i must eventually have constant difference δ_i along a run.

Otherwise, if π' above contains a loop, the difference between the values of x_i and x_{i+1} is not necessarily constant, but lower-bounded by the length δ_i of the loop-free sub path of π'. Thus, in an accepting run, the difference between y_i and y_{i+1} must also be at least δ_i, which is asserted by a y-constraint of the form $y_i - y_{i+1} \geq \delta_i$.

Finally, we describe the reason for which accepting runs can have non-elementary lengths. Imagine that the counter x_i uses a loop in order to catch up with the counter y_i, before the counter y_i is initialised, as described above. Now, suppose that on this loop the counter y_{i-1} has updates of the form `×2+1`. Thus, if there exists an accepting run, and the counter y_i has value m_i in the final configuration of this accepting run, then, the counter x_i must also have value m_i at the end of the run. Thus, the accepting run must contain around m_i loops in order for x_i to catch up to the value of y_i. However, these loops have increase the value of y_{i-1} to an exponential value which implies that there must be another loop before the initialisation of the counter y_{i-1} increasing the value of x_{i-1}. Now, if we imagine that this second loop also contains updates of the type `×2+1` for the counter y_{i-2}, and iterate the above argument, one can see how we can obtain accepting runs of non-elementary lengths.

3.4.2 Abstract configurations for restricted LAVASS

Our decision procedure for non-emptiness of restricted LAVASS is based on reducing this problem to a reachability problem in a carefully designed finite-state abstraction of the state-space of LAVASS. Throughout this section, let $\mathcal{V} = \langle Q, \Sigma, \Delta, \lambda, q_0, F, \Phi \rangle$ be a restricted LAVASS with $2d$ counters. We first define the state space of the abstracted LAVASS.

Definition 19. *An abstract configuration is a tuple*

$$\alpha = (q, m_1, n_1, \ldots, m_d, n_d, u_1, u_2, \ldots, u_{d-1}, \ell_1, \ldots, \ell_d) \\ \in Q \times (\mathbb{N} \cup \{\bot\})^{2d} \times \mathbb{N}^{d-1} \times (\mathbb{N} \cup \{\top\})^d,$$

such that $m_i, n_i \in [0, 2dM_i] \cup \{\bot\}$ *and* $u_i \in [0, U_i]$ *and* $\ell_i \in [0, M_i - 1] \cup \{\top\}$*, where*

- $M_i := \lfloor |Q|^{((1/8)\cdot 32^{i-1}+1)} \rfloor$*; and*
- $U_i := |Q|^{(32^{i-1}+4)}$.

The idea is that m_i, n_i store the residue classes modulo ℓ_i of the counter pair x_i, y_i respectively, where the value $\top$ for ℓ_i acts as an indicator that we are storing actual values and not residue classes. The value $\bot$ for some x_i or y_i indicates that the counter has not yet been initialised, which is different to the counter having value 0 modulo ℓ_i. If for an update function f, $f = \texttt{++}$ or $f = \texttt{×2+1}$ then $f(\bot) := 1$; otherwise $f(\bot) := \bot$, and we stipulate that $\bot \bmod n = \bot$. The value of u_i in an abstract configuration carries the current difference between the value of the counters y_i and y_{i+1}. This difference is potentially unbounded, however for our purposes it suffices to only store its value if it is less than U_i, and to indicate the fact that it is at least U_i by the value $u_i = U_i$.

We denote the (finite) set of all abstract configurations of $\mathcal{V}$ by $A(\mathcal{V})$. Let us now define a transition relation $\dot{\to} \subseteq A(\mathcal{V}) \times \Delta \times A(\mathcal{V})$ such that $\alpha \xrightarrow{t} \alpha'$, $t = (q, a, q') \in \Delta$ if and only if:

- $\alpha = (q, m_1, n_1, \ldots, u_{d-1}, \ell_1, \ldots, \ell_d)$ and $\alpha' = (q', m'_1, n'_1, \ldots, u'_{d-1}, \ell_1, \ldots, \ell_d)$;
- $\lambda(t) = (f_{x_1}, f_{y_1}, \ldots, f_{x_d}, f_{y_d})$;
- $f_{x_i} = \texttt{++}$ for all i such that $m_i \neq \bot$;
- if $\ell_i \neq \top$, $m'_i = f_{x_i}(m_i) \bmod \ell_i$ and $n'_i = f_{y_i}(n_i) \bmod \ell_i$;
- if $\ell_i = \top$, $m'_i = f_{x_i}(m_i)$ and $n'_i = f_{y_i}(n_i)$; and

- for all $i \in \{1, \ldots, d-1\}$,

$$u_i' = \begin{cases} min(2u_i + 1, U_i) & \text{if } f_{y_i} = \texttt{×2+1}, f_{y_{i+1}} = \texttt{×2} \\ min(2u_i, U_i) & \text{if } f_{y_i} = f_{y_{i+1}} \\ min(2u_i - 1, U_i) & \text{if } f_{y_i} = \texttt{×2}, f_{y_{i+1}} = \texttt{×2+1}. \end{cases}$$

Assuming that the value of y_i is at least the value of y_{i+1}, which we will always ensure, the definition of how to update u_i ensures that it exactly stores the difference $y_i - y_{i+1}$ unless the difference becomes too large, in which case it is levelled off at U_i.

An abstract configuration path is a sequence of abstract configurations and transitions of the form $R = \alpha_1 \xrightarrow{t_1} \alpha_2 \xrightarrow{t_2} \cdots \xrightarrow{t_{n-1}} \alpha_n$.

Given two consecutive y-counters y_i, y_{i+1} and $\delta_i \in \mathbb{N}$, we say that $y_i - y_{i+1} = \delta_i$ and $y_i - y_{i+1} \geq \delta_i$ are *y-constraints*. Let Y be a set of y-constraints, an abstract configuration $\alpha = (q, m_1, n_1, \ldots, u_1, \ldots, u_{d-1}, \ell_1, \ldots, \ell_d)$ *respects* Y whenever

- $u_i \geq \delta_i$ for all constraints of type $y_i - y_{i+1} \geq \delta_i$ in Y,
- and $u_i < U_i$ and $u_i = \delta_i$ for all constraints $y_i - y_{i+1} = \delta_i$ in Y.

We say that α_f is a *final abstract configuration respecting* Y whenever $q \in F$, $m_i = n_i$ for all $1 \leq i \leq d$, and α_f respects Y.

3.4.3 Witnessing certificates

While any concrete accepting run of an LAVASS gives rise to an abstract configuration path ending in an accepting abstract configuration, the converse does not hold. This motivates the introduction of reachability and witnessing certificates, which are special abstract configuration paths that carry further information that eventually enables us to derive from a witnessing certificate a concrete accepting run of an LAVASS.

A *reachability certificate* is a tuple (R, X, Y, L) such that $R = \alpha_1 \xrightarrow{t_1} \alpha_2 \xrightarrow{t_2} \cdots \xrightarrow{t_{n-1}} \alpha_n$ is an abstract configuration path, and $X, Y, L \colon \{1, \ldots, d\} \to \{1, \ldots, n\}$. Here, $X(i)$ and $Y(i)$ indicate the position where the x_i-counter and y_i-counter obtain a value different from $\perp$ for the first time. Moreover, $L(i)$ is the position where a loop of length ℓ_i can be found. Formally, (R, X, Y, L) is required to have the following properties:

(a) $\alpha_1 = (q_0, \perp, \ldots, \perp, 0, \ldots, 0, \ell_1, \ldots, \ell_d)$ and if $\ell_i = \top$ then $\ell_j = \top$ for all $i < j \leq d$;

(b) $\lambda(x_i, t_{X(i)-1}) = \texttt{++}$ and $\lambda(y_i, t_{Y(i)-1}) = \texttt{×2+1}$ for all $1 \leq i \leq d$;

(c) $\lambda(x_i, t_j) = \texttt{id}$ for all $1 \leq j < X(i) - 1$;

(d) $\lambda(y_i, t_j) = \texttt{×2}$ for all $1 \leq j < Y(i) - 1$;

(e) X, Y, L are monotonic; and

(f) for all $1 \leq i \leq d$, if $\ell_i \neq \top$ then

- $X(i) \leq L(i) < Y(i)$, and
- there is a simple $\alpha_{L(i)}$-loop $\alpha_{L(i)} \xrightarrow{t'_1} \alpha'_2 \xrightarrow{t'_2} \cdots \xrightarrow{t_{\ell_i - 1}} \alpha'_{\ell_i - 1} \xrightarrow{t'_{\ell_i}} \alpha_{L(i)}$ of length ℓ_i.

Those conditions can be interpreted as follows. Condition (a) asserts that the certificate starts in an initial abstract configuration. We require that $\top$ monotonically propagates since the absence of a loop for counter x_i implies that the remainder of a path is short, hence we can afford to subsequently store actual counter values and not residue classes. Conditions (b), (c) and (d) assert that $X(i)$ and $Y(i)$ are the first position where the counters x_i, y_i hold a value different from $\bot$. Condition (e) states that the counters x_{i+1}, y_{i+1} do not carry a value different from $\bot$ before the counters x_i and y_i, respectively. Condition (f) implies that, if $\ell_i \neq \top$ then between the first update for counter x_i and the first update for counter y_i there is a position $L(i)$ where we can find a loop in the abstract configurations of length ℓ_i. Notice that if $x_j = \bot$ or $y_j = \bot$ in $\alpha_{L(i)}$ then x_j and y_j remain to hold $\bot$ along this loop, i.e., this loop does not update counters that have not been initialised already.

Given R, the set of y-constraints induced by R is the smallest set containing

- $y_i - y_{i+1} \geq \delta_i$, where $\delta_i := X(i+1) - X(i)$ if there is a j such that $X(i) \leq L(j) < X(i+1)$; and
- otherwise $y_i - y_{i+1} = \delta_i$, where $\delta_i := X(i+1) - X(i)$,

for all $1 \leq i < d$ such that $\ell_i \neq \top$.

We introduce some further notation. Given a reachability certificate (R, X, Y, L), we denote by $\pi(R)$ the run corresponding to R in the configuration graph of V, with the initial configuration $(q_0, 0, 0, \ldots, 0, 0)$. Given indices $1 \leq i \leq j \leq n$, we denote by $R[i, j]$ the segment $\alpha_i \xrightarrow{t_i} \alpha_{i+1} \cdots \xrightarrow{t_{j-1}} \alpha_j$ of R, and by $R[i] := \alpha_i$. We say that R is a *witnessing certificate* if, for $a \leq d$ being the largest index such that $\ell_a \neq \top$:

- $R[1, Y(a)]$ is a simple path and $n - Y(a) \leq 2dM_{d+1}$;

- α_n is a final abstract configuration respecting the set of induced y-constraints; and
- $\mathrm{val}(\pi(R), x_a) \leq \mathrm{val}(\pi(R), y_a)$.

Sometimes we will speak of witnessing certificates *restricted* to a set of counters. By that we mean a witnessing certificates where the relevant Conditions (a)–(f) are only required for that set of counters.

In the next section, we prove the following theorem that will enable us to give a proof for Proposition 18.

Theorem 20. *The language of a restricted LAVASS V is non-empty if and only if there exists a witnessing certificate for V.*

3.5 Witnessing certificates witness non-emptiness of restricted LAVASS

In this section, we prove Theorem 20. The proof is split into the two directions. First, we argue that the existence of a witnessing certificate for a LAVASS implies that the language of the LAVASS is non-empty. Subsequently, we show the converse direction.

Witnessing certificates imply non-emptiness of restricted LAVASS

Proposition 21. *If there exists a witnessing certificate for a restricted LAVASS V then $L(V) \neq \emptyset$.*

The idea behind the proof of Proposition 21 is that, from a witnessing certificate (R, X, Y, L) of a LAVASS V, we obtain a sequence of runs of V such that the final run in that sequence is an accepting run of V. Initially, we obtain a run that ends in a configuration where the counters are in a congruence relation. We then carefully pump the simple loops pointed to by L, beginning from the last counter and working towards the first.

Let (R, X, Y, L) be a witnessing certificate, and let $\pi(R)$ be the run in the configuration graph of V induced by R. Let $a \leq d$ be maximal such that $\ell_a \neq \top$. We now define a sequence of runs $\pi_0, \ldots, \pi_a$ such that the following invariant holds. In the final configuration of π_i,

(i) $m_j \leq n_j$ and $m_j \equiv n_j \bmod \ell_j$ for the j-th counter pair, $1 \leq j \leq a - i$; and

(ii) $m_j = n_j$ for the j-th counter pair, $a - i < j \leq d$.

It is clear that π_a then witnesses $L(V) \neq \emptyset$. We proceed by induction on i.

Base case $i = 0$: Let $\pi_0 = \pi(R)$. Since R is a witnessing certificate, $val(\pi(R), x_a) \leq val(\pi(R), y_a)$, and hence $m_a \leq n_a$ in the last configuration of π_0. Moreover, R respects the set of induced y-constraints. Hence $n_{a-1} - n_a \geq \delta_{a-1}$, where δ_{a-1} is the length of the path from $R[X(a-1)]$ to $R[X(a)]$. Hence $n_{a-1} - n_a \geq m_{a-1} - m_a$ and thus $m_{a-1} \leq n_{a-1}$. Iterating this argument for the remaining counters, we get that (i) of the invariant is fulfilled for π_0; (ii) trivially holds since R ends in an accepting abstract configuration.

Induction step $i > 0$: Let π_{i-1} be the path that exists by the induction hypothesis. If $m_{a-i} = n_{a-i}$ in the last configuration of π_{i-1} then we are done and take $\pi_i = \pi_{i-1}$; otherwise $m_{a-i} < n_{a-i}$ and $m_{a-i} \equiv n_{a-i} \bmod \ell_{a-i}$. Hence, there is some $k \in \mathbb{N}$ such that $n_{a-i} = k \cdot \ell_{a-i}$. Since $\ell_i \neq \top$, let $\beta := \alpha_{L(a-i)} \xrightarrow{t_1} \alpha_2 \xrightarrow{t_2} \cdots \alpha_{\ell_i - 1} \xrightarrow{t_{\ell_i}} \alpha_{L(a-i)}$ be the simple α-loop at position $L(a-i)$ that is guaranteed to exist since R is a witnessing certificate. We insert the transitions of β^k and the induced updated configurations into π_{i-1} at position $L(a-i)$. Notice that $L(a-i) < X(a-i+1)$. Otherwise, by the definition of the induced y-constraints, $y_{a-i} - y_{a-i+1} = \delta_{a-i}$ is in the set of induced y-constraints, where $\delta_i = X(a-i+1) - X(a-i)$. Since the last abstract configuration of R respects the set of y-constraints, it must be the case that in the last configuration of π_{i-1}, $n_{a-i} - n_{a-i+1} = \delta_{a-i}$ and $m_{a-i} - m_{a-i+1} = \delta_{a-i}$, so $m_{a-i} = n_{a-i}$, because after the position $X(a-i+1)-1$ in R and thus π_{i-1}, the counters x_{a-i}, x_{a-i+1} get incremented simultaneously. This contradicts our assumption that $m_{a-i} \neq n_{a-i}$. Thus, the counters $x_{a-i+1}, y_{a-i+1}, \ldots, x_d, y_d$ remain unchanged by the insertion of β^k, so (ii) and consequently (i) continues to hold in the last configuration of π_i for those counters. Moreover, due to the ordering conditions imposed on witnessing certificates, the value of y_{a-i} does not change either, and hence $m_{a-i} = n_{a-i}$ in the last configuration of π_i. Since β is a loop in the abstract configuration space, we have $m_j \equiv n_j \bmod \ell_j$ for all $1 \leq j < a - i$ and the values of u_j, for all $1 \leq j < a$ are preserved.

3.5.1 Non-emptiness of restricted LAVASS implies the existence of witnessing certificates

Proposition 22. *If $L(V) \neq \emptyset$ for a restricted LAVASS V then there exists a witnessing certificate for V.*

We begin by defining a function that turns a configuration from $C(V)$ into an

abstract configuration. This function is parameterised by $\ell_1, \ldots, \ell_d \in \mathbb{N}_+ \cup \{\top\}$:

$$f_V((q, m_1, n_1, \ldots, m_d, n_d), \ell_1, \ldots, \ell_d) := (q, m_1 \circ \ell_1, n_1 \circ \ell_1, \ldots, m_d \circ \ell_d, n_d \circ \ell_d, \\ min(n_1 - n_2, U_1), \ldots, min(n_{d-1} - n_d, U_{d-1}), \ell_1, \ldots, \ell_d)\,.$$

Here, $m \circ \ell := \bot$ if $m = 0$; $m \circ \ell := m \bmod \ell$ if $\ell \in \mathbb{N}_+$; and $m \circ \ell := m$ if $\ell = \top$. We lift the definition of f_V to concrete runs π in the natural way, and write $f_V(\pi, \ell_1, \ldots, \ell_d)$ for the resulting sequence of abstract configurations. Let $\pi = c_1 \xrightarrow{t_1} c_2 \cdots \xrightarrow{t_{n-1}} c_n$ be a run witnessing $L(V) \neq \emptyset$. We show how to obtain a witnessing certificate R from π. Without loss of generality, in c_n we have $m_1 \geq m_2 \geq \ldots m_d > 0$.

To this end, we show how from the accepting run π we can iteratively define a sequence $R_0, R_1, R_2 \ldots, R_d$ of abstract runs and identify the required $\ell_1, \ldots, \ell_d \in \mathbb{N}_+ \cup \{\top\}$ and X, Y, L such that (R_d, X, Y, L) is a reachability certificate. Let $X(i) := j$ such that j is the first position in π where the value of counter x_i is non-zero; analogously define $Y(i)$ to be the first position where the value of y_i is non-zero. Clearly, X, Y are monotonic and $X(i) \leq Y(i)$, for all $1 \leq i \leq d$. Otherwise, if a counter y_i gets initialised before the counter x_i in π, it must be the case that $n_i > m_i$ in c_n and therefore c_n cannot be an accepting configuration.

Recall that π has length n. In our proof, the subsequent technical lemma will allow us to conclude that, if for a counter pair x_i, y_i the y_i counter gets updated shortly after the x_i counter then the run will end shortly after and counter pairs x_j, y_j for $j \geq i$ will consequently have small values.

Lemma 23. *If $Y(i) - X(i) \leq dM_i$ for some $1 \leq i \leq d$ then $n - Y(i) < dM_i$, so $m_j, n_j \leq 2dM_i$ in c_n for all $i \leq j \leq d$.*

Proof. We have that $Y(i) - X(i) \leq dM_i$ implies that $val(\pi[1, Y(i)], x_i) \leq dM_i + 1$, and since $val(\pi[1, Y(i) + k], y_i) \geq 2^k$ we get that:

- $val(\pi, y_i) \geq 2^{n - Y(i)}$; and
- $val(\pi, x_i) \leq dM_i + n - Y(i) + 1$.

Assume that $n - Y(i) \geq dM_i$. Then, $2^{n-Y(i)} - (dM_i + n - Y(i) + 1) > 2^{n-Y(i)} - (2n - 2Y(i) + 1) > 0$, if $n - Y(i) \geq 3$. However, π is an accepting path, so $val(\pi, x_i) = val(\pi, y_i)$, and we get a contradiction. Thus, we must have that $n - Y(i) < dM_i$ which implies that $val(\pi, x_i) \leq 2dM_i$, so $m_i = n_i \leq 2dM_i$ and for any j, $i < j \leq d$, $m_j \leq m_i$ and $n_j \leq n_i$, so $m_j, n_j < 2M_i$ in c_n, for all $i \leq j \leq d$ since π is an accepting path. □

Let $R_0 := f_V(\pi, 1, 1, \ldots, 1)$. Note that R_0 together with X and Y as defined above adhere to Conditions (a)–(e) of reachability certificates.

Suppose R_{i-1} and $\ell_1, \ldots, \ell_{i-1}$ have been constructed. If $i > 1$, and $L(i-1) \geq X(i)$ or $\ell_{i-1} = \top$ then we choose $\ell_i := \ell_{i-1}$, $L(i) = L(i-1)$ and $R_i := f_V(\pi, \ell_1, \ldots, \ell_i, 1, \ldots, 1)$. Otherwise, we distinguish two cases.

- $Y(i) - X(i) < dM_i$: we choose $\ell_i := \top$ and $L(i) := X(i)$.
- $Y(i) - X(i) \geq dM_i$: then there is a segment in $R_{i-1}[X(i), Y(i)]$ of length greater than M_i on which no x-counter has its first ++ transition. Let N_i be the number of different abstract configurations on this segment. Since $\ell_{i-1} \neq \top$ we know that m_j, n_j can take at most M_j different values for all $1 \leq j < i$, as they can either be $\bot$ or a residue class modulo M_j. Also, for all $i \leq j \leq d$ the values of m_j, n_j have a constant value, either 0 or $\bot$, on this segment, and $u_i = \cdots = u_d = 0$ in all abstract configurations of this segment. So

$$\begin{aligned} N_i &\leq |Q| \prod_{1 \leq j < i} M_j^2 \cdot U_j \\ &\leq |Q| \prod_{1 \leq j < i} |Q|^{(1/4)\cdot 32^{j-1}+2+32^{j-1}+4} \\ &\leq |Q|^{(1/3968)(5\cdot 32^i - 23968)+6i+1} \\ &< |Q|^{(1/8)\cdot 32^{i-1}+1} \\ &= M_i \end{aligned}$$

  By the pigeonhole principle, there is a smallest k, $X(i) \leq k < Y(i)$, $\ell < M_i$, and a simple loop $\alpha_k \xrightarrow{t_k} \cdots \xrightarrow{t_{k+\ell}} \alpha_{k+\ell+1} = \alpha_k$ in R_{i-1}. We choose $L(i) := k$, $\ell_i := \ell$ and let $R_i := f_V(\pi, \ell_1, \ldots, \ell_i, 1, \ldots, 1)$.

By construction, (R_d, X, Y, L) is a reachability certificate. It remains to turn it into a witnessing certificate. In particular, this requires to remove loops from R_d, to ensure that the final segment of R_d is short, and to establish that R_d is consistent with the implied y-constraints.

Let $R := R_d = f_V(\pi, \ell_1, \ldots, \ell_d)$ and a be the largest index such that $\ell_a \neq \top$. In order to make R loop-free, we iterate the following process:

- Identify the first simple loop $\alpha_k \xrightarrow{t_k} \cdots \xrightarrow{t_{k+\ell}} \alpha_{k+\ell+1}$ in $R[1, Y(a)]$ and replace it by α_k; observe that for $I := \{k+1, \ldots, k+\ell\}$, we have $I \cap \{X(i), Y(i), L(i) : 1 \leq i \leq d\} = \emptyset$ since $\alpha_{X(i)-1} \xrightarrow{t_{X(i)-1}} \alpha_{X(i)}$ occurring in R_d means that x_i has value $\bot$ in $\alpha_{X(i)-1}$ and a value different from $\bot$ in $\alpha_{X(i)}$, and thus $\alpha_{X(i)}$ cannot

be part of a loop; the same argument applies to any $Y(i)$. Finally, since $L(i)$ was chosen as the index of the first configuration of the first cycle appearing after $X(i)$, we have $L(i) \notin I$ for all $1 \leq i \leq d$ as well.

- Update X, Y, L such that for all i such that $X(i) > k$, $X(i) := X(i) - \ell$, and analogously $Y(i) := Y(i) - \ell$ and $L(i) := L(i) - \ell$ for the respective i.

This process guarantees that $R[1, Y(a)]$ is loop-free. It is easy to verify that (R, X, Y, L) obtained in this way is a reachability certificate and that the last abstract configuration of R is accepting.

We now show that the y-constraints induced by R are valid in the final configuration of R. To this end, we first show that for all $1 \leq i \leq d$ such that $\ell_i \neq \top$, $X(i+1) - X(i) < U_i$. Consider the simple path $\alpha_{X(i)} \xrightarrow{t_{X(i)}} \alpha_{X(i)+1} \xrightarrow{t_{X(i)+1}} \cdots \xrightarrow{t_{X(i+1)-1}} \alpha_{X(i+1)}$. If $Y(i) \geq X(i+1)$ then clearly $X(i+1) - X(i) \leq N_i < U_i$, where N_i is defined as above. Otherwise, there is a $k \in \mathbb{N}$ such that the path decomposes as

$$\alpha_{X(i)} \xrightarrow{t_{X(i)}} \cdots \alpha_{Y(i)} \xrightarrow{t_{Y(i)}} \cdots \alpha_{Y(i)+k} \xrightarrow{t_{Y(i)+k}} \cdots \xrightarrow{t_{X(i+1)-1}} \alpha_{X(i+1)}$$

and

- $u_i = 0$ in all abstract states α_j with $X(i) \leq j \leq Y(i)$;
- $u_i = U_i$ in all abstract states α_j with $Y(i) + k \leq j \leq X(i+1)$; and
- $k \leq \log U_i$.

Thus, the maximum length of $R[X(i), X(i+1)]$ is bounded by:

$$\begin{aligned} & N_i \cdot M_i + \log U_i + N_i \cdot 2M_i \\ \leq\ & 2 \cdot M_i^3 + \log U_i \\ \leq\ & |Q|^{(3/8)\cdot 32^{i-1}+4} + |Q|^{5(i-1)+1} + 4\,|Q| \\ <\ & |Q|^{32^{i-1}+4} \\ =\ & U_i \end{aligned}$$

We can now show that R respects the induced y-constraints. Fix some $1 \leq i \leq d$ such that $\ell_i \neq \top$. We distinguish two cases.

- There is no $1 \leq j \leq a$ such that $X(i) \leq L(j) \leq X(i+1)$. Thus, we know that $y_i - y_{i+1} = \delta_i$ is in the set of induced y-constraints. Also, $val(\pi, y_i) - val(\pi, y_{i+1}) = val(\pi, x_i) - val(\pi, x_{i+1}) = X(i+1) - X(i) = \delta_i$ since we did not

remove any abstract loops on the segment of R_d between the first ++ update for x_i and the first ++ update for x_{i+1}. Finally, since $\delta_i < U_i$ by the above argument, we conclude that $u_i = \delta_i$ in the last abstract configuration $R[n]$ of R.

- Otherwise, $X(i) \leq L(i) \leq Y(i)$, so $y_i - y_{i+1} \geq \delta_i$ is in the set of induced y-constraints. However, $val(\pi, y_i) - val(\pi, y_{i+1}) = val(\pi, x_i) - val(\pi, x_{i+1}) \geq X(i+1) - X(i) = \delta_i$ and again because $\delta_i < U_i$ we can conclude that $u_i \geq \delta_i$ in $R[n]$.

This establishes that the y-constraints are satisfied. Let n be the index of the last abstract configuration of R. For the final step, we now argue that $val(\pi(R), x_a) \leq val(\pi(R), y_a)$ and $n - Y(a) \leq 2dM_{d+1}$. We make a case distinction:

- $a = d$: Note that $val(\pi, x_d) = val(\pi, y_d)$. Since we only remove loops from $R_d[1, Y(d)]$, we have that $val(\pi(R), x_d) \leq val(\pi(R), y_d)$. If $n - Y(d) \leq 2dM_{d+1}$ we are done with (R, X, Y, L) as a witnessing certificate. Otherwise, assume $n - Y(d) > 2dM_{d+1}$. This implies that the path $R[Y(d), n]$ must contain at least one simple loop. Consider iterating the following process:
  - remove the first simple loop from $R[Y(d), n]$ and update $n := n - \ell$, where ℓ is the length of the simple loop that was removed; and
  - stop if $n - Y(d) \leq 2dM_{d+1}$.

  We argue that, $n - Y(d) \geq M_d^2$. Let R' and n' be the previous values of R, n before the last iteration. It must be that $n' > 2dM_{d+1}$ and since the length of any simple loop of $R'[Y(d)+1, n']$ is bounded by M_{d+1}, we get that $n - Y(a) \geq M_{d+1} \geq M_d^2$. Note that $Y(d) - X(d) \leq M_d \cdot |Q| \cdot \prod_{1 \leq j < d} M_j^2 \cdot U_j \leq M_d^2$, so $val(\pi(R[1, Y(d)]), x_d) \leq M_d^2$. It must be then the case that $val(\pi(R), x_d) \leq val(\pi(R), y_d)$.

- $a < d$: we know $n - Y(a) \leq 2dM_{d+1}$ by Lemma 23. Moreover, we must have that $val(\pi(R), x_a) \leq val(\pi(R), y_a)$ since $val(\pi, x_a) = val(\pi, y_a)$ and we do not remove loops after the counter y_a is incremented.

This concludes our proof of Proposition 22 and together with Proposition 21, we conclude the proof of Theorem 20.

3.6 Discussion

The main result of this chapter has been to show that the existential theory of generalised Semënov arithmetic is decidable in EXPSPACE. On a technical level, this result was obtained by showing that a restricted class of labelled affine VASS has an EXPSPACE-decidable language emptiness problem. The structural restrictions imposed on those restricted LAVASS are rather strong, though necessary to obtain a decidable class of LAVASS.

An interesting aspect of our approach is that it establishes a special type of automaticity of the existential fragment of a logical theory that is different from traditional notions of automaticity, which are based on finite-state automata or tree automata over finite or infinite words and trees [16, 53], respectively. It would be interesting to better understand whether there are natural logical theories whose (existential) fragments are, say, Petri-net or visibly-pushdown automatic.

Another direction worth investigating would be to better understand the consequences of our restrictions. All together they give us an EXPSPACE procedure, but we do not fully understand what happens if we only use a subset of them. It seems that separating counters into unary ones (x-counters) and binary ones (y-counters) is crucial for decidability, and so is the restriction that the binary counters need to grow continuously once initialised. However, it is unclear if this second restriction is necessary for the unary counters which are less powerful. Dropping this condition would enable us to model second-order logics with counting where the binary words that the LAVASS reads correspond to sets and the unary counters would then count the size of these sets.

We have ignored algorithmic lower bounds throughout this chapter, but it would, of course, be interesting to see whether the upper bounds of the decision problems we considered in this article are tight. It is clear that generalised Semënov arithmetic is PSPACE-hard since it can readily express the DFA intersection non-emptiness problem, but this still leaves a considerable gap with respect to the EXPSPACE upper bound we established. In particular, the recent results of [12] showing an NEXP upper bound for the existential fragment of Semënov arithmetic suggest that, if an EXPSPACE lower bound for existential generalised Semënov arithmetic is possible, it will require the use of regular predicates.

As an application of our EXPSPACE upper bound for existential generalised Semënov arithmetic, it was shown in [33] that a certain class of string constraints with length constraints is decidable in EXPSPACE. This class allows for existentially

quantifying over bit-strings, and to assert that the value of a string variable lies in a regular language, as well as Presburger-definable constraints over the lengths of the bit-strings stored in string variables and the numerical values of those variables (when viewed as encoding a number in binary). Decidability of this class was left open in [13]. We settle this open problem by showing that it can be reduced to the existential fragment of generalised Semënov arithmetic.

Chapter 4

Bounded Arithmetic and VASS

In this chapter, similarly to the previous one, we study the connection between vector addition systems with states (VASS) and two different arithmetic theories (Presburger arithmetic and Rudimentary arithmetic). This connection enables us to obtain new results related to the reachability problem for VASS, in the setting where the VASS and the final configuration are fixed and only the initial configuration is variable. The chapter starts by introducing the concept of *counter programs*. This is a new formalism presented e.g., in [25], that allows for presenting VASS in a serialised way. This formalism is not specific to VASS, as it can model general counter automata, but we only use this formalism in this chapter, for reasons that we describe in the next section.

Then, we introduce the validity problem for short Presburger arithmetic. For this problem, we consider a formula Φ of Presburger arithmetic containing a Boolean combination of inequalities of the form $\boldsymbol{a}\boldsymbol{x} \leq b$, for which the structure of the formula is fixed and the only inputs to the problem are the coefficients $\boldsymbol{a}$ and b in the inequalities. A few year ago, it was shown in [71] that this problem is hard for every level of the polynomial hierarchy. Next, we present a reduction from this problem to the reachability problem for a fixed VASS. This gives us a first lower bound for fixed VASS reachability in the case where the inputs are presented in binary. Later on, by a series of reductions, we can establish a PSPACE lower bound, although the proof is less informative.

We also study the setting in which the inputs are presented in unary. This turns out to be a very delicate issue. As a first step, we establish a tight correspondence between the reachability in VASS and the first-order theory of initial segments of $\mathbb{N}$ with the arithmetic relations of addition and multiplication, and counting quantifiers, $\mathbf{FOunC}(+, \times)$. Given a fixed rudimentary relation $\Phi(x_1, \ldots, x_k)$, we show how to construct a fixed VASS $\mathcal{V}$ and fixed polynomials $p_1, \ldots, p_m$ such that

$\Phi(n_1, \ldots, n_k)$ evaluates to true in $\underline{N}$ if and only if there is a run in $\mathcal{V}$ starting in $(p_1(N, n_1, \ldots, n_k), \ldots, p_m(N, n_1, \ldots, n_k))$ and ending in the zero vector. It thus follows that reachability in fixed VASS under unary encoding of configurations is at least as hard as evaluating any rudimentary relation under unary encoding of numbers. Hence, reachability queries in fixed VASS can, e.g., determine primality and square-freeness of a number given in unary.

Finally, we provide a PSPACE hardness result for the VASS reachability problem when the VASS is fixed and the inputs are presented in binary. This is achieved via a series of standard reductions and we include it in this chapter since the complexity bound is tighter than the one obtained via the connection with short Presburger.

4.1 Related Work

To the best of our knowledge, the reachability problem for fixed VASS has not yet been systematically explored. Closest to the topics of this paper is the work by Rosier and Yen [76], who conducted a multi-parameter analysis of the complexity of the boundedness and coverability problems for VASS.

However, the study of the computation power of other fixed machines has a long history in the theory of computation. The two classical decision problems for a computation model are *membership* (also called the *word problem*) and *reachability*. Membership asks whether a given machine accepts a given input; the (generic) reachability problem asks whether given an initial and a target configuration, there is a path in the transition system induced by a given machine from the initial configuration to the target configuration. The most prominent example of a reachability problem is the halting problem for different kinds of machines. Classically, the computational complexity of such problems assumes that both the computational model and its input word (for membership) or configurations (for reachability) are part of the input. However, these are two separate parameters. For example, in database theory, the database size and the query size are often considered separately, since the complexity of algorithms may depend very differently on these two parameters, and the sizes of these two parameters in applications can also vary a lot [86]. One approach to study such phenomena is to fix either the database or the query. More generally, the field of parameterised complexity studies the computational difficulty of a problem with respect to multiple parameters of the input.

Returning to our setting, this means fixing either the machine or its input. In this thesis, we concentrate on the former. The question can then be seen as follows:

in relation to a problem such as membership or reachability, which machine is the hardest one in the given computation model? For some models, the answer easily follows from the existence of universal machines, i.e., machines which are able to simulate any other machine from their class. A classical example here is a universal Turing machine. Sometimes the ability to simulate all other machines has to be relaxed, for example as for Greibach's hardest context-free language [43]. Greibach showed that there exists a fixed context-free grammar such that a membership query for any other context-free grammar can be efficiently reduced to a membership query for this grammar. Similar results are known for two-way non-deterministic pushdown languages [22, 77].

4.2 Counter programs

We start by introducing counter programs. A counter program is a primitive imperative program that executes arithmetic operations on a finite number of counter variables. Formally, a *counter program* consists of a finite set $\mathcal{X}$ of global counter variables (called *counters* subsequently for brevity) ranging over the natural numbers, and a finite sequence $1, \ldots, m$ of line numbers (subsequently *lines* for brevity), each associated with an instruction manipulating the values of the counters or a control flow operation. Each instruction is in of one the following forms:

- $x \mathrel{+}= c$ (increment counter x by constant $c \in \mathbb{N}$),
- $x \mathrel{-}= c$ (decrement counter x by constant $c \in \mathbb{N}$),
- **goto** L_1 **or** L_2 (non-deterministically jump to the instruction labelled by L_1 or L_2),
- **skip** (no operation).

We write **goto** L as an abbreviation for **goto** L **or** L, and also allow statements of the form **goto** L_1 **or** L_2 **or** $\ldots$ **or** L_k. Moreover, the line with the largest number is a special instruction **halt**. In our examples of counter programs, we usually omit this last line if it is not referenced explicitly. Before we describe the semantics of a counter program, we show an example of a counter program. In Figure 4.1, we present a counter program that uses a single counter x and consists of five lines. Starting in line 1, the program non-deterministically loops and decrements the counter x by three every time, until it increments x by one and terminates. One can see that the VASS on the right has the same behaviour. Also, we can easily model more complex

arithmetic, or let the counters interact by allowing instructions of the type $x = 2x+5$, or $x = y - 3$.

1: **goto** 2 **or** 4
2: x −= 3
3: **goto** 1
4: x += 1
5: **halt**

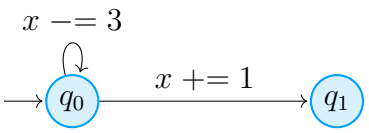


Figure 4.1: A counter program and its coresponding VASS

Before we define the semantics of a counter program, we introduce the syntactic operation of *substitution* which, substitutes a given line of a counter program (which we always assume to have a **skip** instruction) with the instructions of another counter program. This allows us to compose multiple counter programs. Formally, let C_1, C_2 be counter programs with m_1 and m_2 lines respectively. The result of substituting line k, $1 \leq k \leq m_1 - 1$, of C_1 with C_2 is a counter program C_1' with $m_1 + m_2 - 1$ lines obtained, intuitively, by calling C_2 as a sub-routine in this line and when it halts returning control back to C_1. Formally, the instruction corresponding to a line L, $1 \leq L < m_1 + m_2$, is defined as follows:

- if $L < k$, it is the instruction of line L in C_1,
- if $k \leq L < m_2 + k - 1$, it is the instruction of line $L - k + 1$ in C_2,
- if $L = m_2 + k - 1$, it is the instruction **skip**,
- if $m_2 + k \leq L$, it is the instruction of line $L - m_2$ in C_1.

The line numbers in **goto** instructions are changed accordingly. We also consider a substitution of several counter programs. When specifying counter programs, to denote substitution of another counter program we just write its name instead of an instruction in a line. Also, we write $C_1; C_2$ for

1: C_1
2: C_2

and C_1 **or** C_2 as syntactic sugar for the counter program:

1: **goto** 2 **or** 4
2: C_1
3: **goto** 5
4: C_2

When C is a counter program, we write **loop** C as an abbreviation for the counter program

1: **goto** 2 **or** 4
2: C
3: **goto** 1

Hence, the counter program in Figure 4.1 corresponds to

1: **loop**
2: $\quad x -= 3$
3: $x \mathrel{+}= 1$

We use indentation to mark the scope of the **loop** instruction. We also assume that if several instructions share the same line and are separated by a semicolon, they all belong to the scope of a loop.

4.2.1 Semantics of counter programs

Exactly as in the case of VASS, a *configuration* of a counter program is an element $(L, f) \in \mathbb{N} \times \mathbb{N}^{\mathcal{X}}$, where $L \in \mathbb{N}$ is a program line with a corresponding instruction, and $f \colon \mathcal{X} \to \mathbb{N}$ is a counter valuation. The semantics of counter programs are defined in a natural way: after executing the instructions on the line L, we either non-deterministically go to one of the specified lines (if the instruction on line L is a **goto** instruction), or, otherwise, we go to the line $L + 1$. After executing the last line, we stop.

One can view a counter program as a VASS by treating line numbers as states and defining transitions as specified by the counter program, each labelled with the respective instruction. It is also easy to see how to convert a VASS into a counter program.

A *run* of a counter program is a sequence $\varrho \colon (L_1, f_1) \to (L_2, f_2) \to \ldots \to (L_n, f_n)$ of configurations defined naturally according to the described semantics. For example, $(1, \{x \mapsto 7\}) \to (4, \{x \mapsto 7\}) \to (5, \{x \mapsto 8\})$ is a run of the counter program in Figure 4.1. Given a run $\varrho \colon (L_1, f_1) \to (L_2, f_2) \to \ldots \to (L_n, f_n)$, we say that ϱ is *terminating* if $L_1 = 1$ and the instruction on line L_n is **halt**, and *zero-terminating* if additionally $f_n(x) = 0$ for all $x \in \mathcal{X}$. We denote by $\mathrm{val}_{\mathrm{end}}(\varrho, x) := f_n(x)$ the value of the counter x at the end of a terminating run. Sometimes, we also want to talk about the value of a counter at a specific point during the execution of a run and define $\mathrm{val}_i(\varrho, x)$ to be the value of the counter x right before we execute the instruction on line i in the run ϱ for the first time, i.e. $\mathrm{val}_i(\varrho, x) := f_k(x)$, where k is the smallest

index such that $L_k = i$. For instance, in the example above, we have $\text{val}_{\text{end}}(\varrho, x) = 8$ and $\text{val}_4(\varrho, x) = 7$. We often construct counter programs that admit exactly one run ϱ from a given initial configuration to a target configuration. In such a setting, we may omit the reference to ϱ and simply write $\text{val}_{\text{end}}(x)$ and $\text{val}_i(x)$. The effect $\text{eff}(\varrho)\colon \mathcal{X} \to \mathbb{Z}$ of a run ϱ starting in $(1, f_1)$ and ending in (n, f_n) is a map such that $\text{eff}(\varrho, x) = f_n(x) - f_1(x)$ for all $x \in \mathcal{X}$.

Recall that we defined the VASS reachability problem (*Problem* 4) as deciding if a VASS $\mathcal{V}$ admits a run from configuration c to configuration c', where $\mathcal{V}, c, c'$ from the input of the problem. We can also define the VASS zero-reachability problem as the problem where we fix c' to be a configuration of the type $(q, \mathbf{0})$. Now, we can define an equivalent problem in the language of counter programs.

Problem 24. Counter program zero-reachability
Input: *A counter program $\mathcal{C}$ and a vector $\boldsymbol{x} \in \mathbb{N}^d$ of initial values of the counters.*
Output: *YES if and only $\mathcal{C}$ has a zero-terminating run from $\boldsymbol{x}$.*

4.2.2 Extending counter programs

As stated previously, one can easily extend the instruction set of counter programs to allow for more complex arithmetic operations on the values of the counters. This would be a first step towards allowing counter programs to represent more expressive formal models such as LAVASS or Cost Register Automata. A major difference between VASS and these former models is that the transitions of VASS are not labelled. It is not clear how one might deal with labelled transitions in a counter program. One option would be to introduce some read instructions, but this would most likely make the formalism cumbersome and hard to work with. Recall however, that in the case of LAVASS, the problem that we study here, language non-emptiness, asks whether or not there exists an accepting path. Thus, one can easily ignore the labels of the transitions and we could rephrase the question in the language of counter programs. This will also be the case for the next chapter when we look at CRA.

We briefly explain why we only use the formalism of counter programs in this chapter, even though we claim that it can be extended to LAVASS and CRA. In the language of counter programs, we get as input the "code" of a counter program, and we are interested in finding out if the programs admits a specific run or not. This is very similar to the classical setting of formal verification, where we have some piece of code and we want to verify, for example, that there are no "bad" runs. In the case of counter programs, the complexity comes from the non-deterministic behaviour

generated by the **goto** instructions, that lead to a structure of runs that is very complicated and hard to analyse. That is why, in formal verification, we generally use formal models as tools for the task of analysing programs, mainly because analysing runs in a program is somehow harder than analysing the coresponding runs in a formal model. Pumping arguments, as the one used in the proof of Theorem 20, work better on transition systems where we can talk about loops and the shape of a run. This is one reason for which we designed formal methods: to have a better language for lines of code. Now, when the task is to analyse properties of formal models, it might seem strange that we go back to lines of code in the shape of counter programs. The answer is the following. When we want to design an algorithm that finds these specific runs (when we are looking for upper bounds for a decision problem), we use the standard formalism because reasoning on counter programs would probably be harder. However, in this chapter, we are looking for lower bounds, so we want to have a VASS that computes something hard. Here, it is nicer to work with counter programs, because when we want to compute things we generally prefer to use code, rather than transition systems.

As a final note, in order to model Counter Automata with counter programs, we also need an instruction that can test counters for zero. This is straightforward, but we do not wish to study the properties of full Counter Automata since most of them are known to be undecidable. However, without instructions that can test the values of counters, it is very hard to have counter programs that can in a sense compute something meaningful. This is why, in the next subsection we describe how we can recover this ability for counter programs, but in a limited way.

4.2.3 Implementation of zero tests

One of the common ways to deal with the issue of simulating zero tests with a counter program is to introduce some restricted zero tests, that is, some gadgets that guarantee that if a run reaches a certain configuration, then along this run, the values of some counters are zero at prescribed positions. In this subsection, summarising [25], we describe such a gadget in the case where the values of counters are bounded by a given number. The number of zero tests that can be performed this way is also bounded. For a counter v, we call this gadget **zero-test**(v), and later on we will use it as a single instruction to test that the value of v is zero before executing it.

Let $N \in \mathbb{N}$ be an upper bound on the value of a counter v. Then, we can introduce a counter $\hat{v}$ and enforce the invariant $f(v) + f(\hat{v}) = N$ to hold in all the configurations of any run of our counter program. We achieve this by ensuring that

every line containing an instruction of type $v \mathrel{+}= c$ must be followed by a line with a $\hat{v} \mathrel{-}= c$ instruction.

We introduce auxiliary counters u_1, u_2 that will be tested for zero only in the final configuration, and hence have no hat counterpart. In the following, the instruction **zero-test**(v) denotes the following gadget:

1: **loop**
2: $\quad v \mathrel{+}= 1; \hat{v} \mathrel{-}= 1; u_2 \mathrel{-}= 1$
3: **loop**
4: $\quad v \mathrel{-}= 1; \hat{v} \mathrel{+}= 1; u_2 \mathrel{-}= 1$
5: $u_1 \mathrel{-}= 2$

Consider an initial configuration in which $f(u_1) = 2n$ and $f(u_2) = 2n \cdot N$ for some $n > 0$. Initially, it is true that $f(u_2) = f(u_1) \cdot N$.

Lemma 25 ([25]). *There exists a run of the counter program* ***zero-test***(v) *that starts in a configuration with $f(u_2) \geq 2$, $f(u_2) = f(u_1) \cdot N$, and ends in a configuration with $f(u_2) = f(u_1) \cdot N$ if and only if $f(v) = 0$ in the initial configuration.*

Proof. The invariant $f(v) + f(\hat{v}) = N$ ensures that the loops on line 1 and line 3 can each decrease the value of u_2 by at most N. Moreover, this can only happen if $f(v) = 0$ in the initial configuration. □

From a configuration with $f(u_2) = f(u_1) \cdot N$, a run that is "incorrectly" executing the **zero-test**(v) subroutine can only reach a configuration with $f(u_2) > f(u_1) \cdot N$. Observe that from such a configuration, we can never reach a configuration respecting the invariant $f(u_2) = f(u_1) \cdot N$ if the values of u_1, u_2 are only changed by **zero-test**(v) instructions. Now, consider a counter v and a counter program C that modifies the values of counters u_1 and u_2 only through the **zero-test**(v) instruction. If we start in a configuration in which $f(u_1) = 2n$ and $f(u_2) = 2n \cdot N$ for some $n > 0$, and we are guaranteed that any run of C cannot execute more than n **zero-test**(v) instructions, then after any run of C, we have that $f(u_2) = f(u_1) \cdot N$ only if the value of the counter v was zero at the beginning of every **zero-test**(v) instruction. If all the counters that we are interested in are bounded by the same value N, we can use a single pair of counters u_1, u_2 to perform zero tests on all our counters. We subsequently call the counters u_1 and u_2 *testing counters*. To summarise, using this technique, we can perform n zero tests on counters bounded by N via a reachability query in a VASS.

Given a configuration (L, f), we say that (L, f) is a *valid configuration* if f respects the condition that $f(u_2) = f(u_1) \cdot N$. A *valid run* is a run that starts in a valid

configuration and ends in a valid configuration. Also, a counter program *admits* a valid run if there exists a valid run that reaches the terminal instruction **halt**. Observe that in every valid run the **zero-test**() subroutine does not change the value of the counter which is tested for zero, that is, this value remains zero. Only the values of the testing counters are changed.

From now on, we make the convention that the instruction $v \mathrel{+}= c$ is an abbreviation for $v \mathrel{+}= c; \hat{v} \mathrel{-}= c$. This allows us to remove the hatted counters from our future counter programs whenever it is convenient for us, which will ease readability. So, if we choose an initial configuration in which $f(v) + f(\hat{v}) = N$, we have that this invariant holds whenever the zero-test gadget is invoked.

We now introduce components. Informally, a component is a counter program acting as a subroutine such that, if it is invoked in a configuration fulfilling the invariants required for valid runs, upon returning, those invariants still hold. Formally, a *component* is a counter program such that:

- there is a polynomial p such that every valid run performs at most $p(N)$ calls of **zero-test**() on all counters, where N is the bound on the counters; and
- the values of u_1 and u_2 are updated only by **zero-test**() instructions.

We conclude this section with Lemma 26, which states that sequential composition and non-deterministic branching of components yields components. We will subsequently implicitly make use of this obvious lemma without referring to it.

Lemma 26. *If C_1, C_2 are components then both $C_1; C_2$ and C_1 **or** C_2 are also components.*

4.3 Connection to arithmetic theories

In this section we describe the connection between VASS, presented as counter programs, and arithmetic theories. As noted, a counter program only has access to zero tests for bounded counters. Thus, we can start by trying to model the rudimentary arithmetic presented in the Preliminaries, where the universe is an initial segment of $\mathbb{N}$. The goal is that given a formula of Φ of $\mathbf{FO}k\text{-}\mathbf{aryC}(+,\times)$, a bound N, and some input values $\boldsymbol{x} \in \underline{N}^n$, one can design a counter program such that $\langle N, +, \times\rangle \models \Phi$ if and only if the counter program has a zero-terminating run from $\boldsymbol{x}'$, where $\boldsymbol{x}'$ depends on $\boldsymbol{x}$ in a trivial way. The counter program aims to "implement" the computations inside Φ. This gives us some new insights into the expressive power of counter programs and also a complexity lower bound.

It is interesting that in a specific setting, we do not need to restrict ourself to rudimentary arithmetic. We can also model a fragment of Presburger arithmetic. This fragment has an infinite universe, but thanks to a clever technique, we can reduce the search space to a finite domain.

4.3.1 Short Presburger arithmetic

As presented in the Preliminaries, Presburger arithmetic is the first-order theory of the structure $\langle \mathbb{Z}, 0, 1, +, < \rangle$. We follow [71] for the definition of short Presburger arithmetic. The class of *short Presburger sentences* with m quantifier alternations (Short-PA$_m$) are formulas of Presburger arithmetic of the form

$$\Psi \equiv \exists \boldsymbol{x}_1 \forall \boldsymbol{x}_2 \cdots Q_m \boldsymbol{x}_m \, \Phi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m),$$

with a fixed number of m quantifier alternations, and where each vector of variables $\boldsymbol{x}_i$ has fixed dimension n_i, and Φ is a fixed positive Boolean combination of linear inequalities of the form

$$a_1 \cdot x_1 + \cdots + a_m \cdot x_m \leq b \,.$$

In other words, everything in Short-PA$_m$ is fixed, except for the coefficients $a_1, \ldots, a_m, b \in \mathbb{Z}$. Denote by $|\Psi|$ the size of Ψ, which is the number of symbols required to write down Ψ assuming binary encoding of numbers.

Theorem 27 ([71]). *Deciding a sentence $\exists \boldsymbol{x}_1 \forall \boldsymbol{x}_2 \cdots Q_m \boldsymbol{x}_{m+2} \, \Phi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{m+2})$ of Short-PA$_{m+2}$ is Σ_m^{P}-complete. The lower bound already holds when Φ consists of at most $10m$ inequalities in $4m+1$ variables $\boldsymbol{x}_1 \in \mathbb{Z}$, $\boldsymbol{x}_2, \boldsymbol{x}_{m+2} \in \mathbb{Z}^2$ and $\boldsymbol{x}_3, \ldots, \boldsymbol{x}_{m+1} \in \mathbb{Z}^4$.*

For the upper bound, it is well known that Presburger arithmetic relativises in the sense that, in order to decide a sentence of Presburger arithmetic, it suffices to only consider numbers in bounded intervals. The following proposition is a consequence of [91, Thm. 2.2].

Proposition 28 ([91]). *Let $\Psi \equiv \exists \boldsymbol{x}_1 \forall \boldsymbol{x}_2 \cdots Q_m \boldsymbol{x}_m \, \Phi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$ be a sentence of Short-PA$_m$. Then there are polynomial-time computable constants $c_1, \ldots, c_m \in \mathbb{N}$ that are exponential in $|\Psi|$ such that Ψ is valid if and only if*

$$\exists \boldsymbol{x}_1 \in [-c_1, c_1]^{n_1} \, \forall \boldsymbol{x}_2 \in [-c_2, c_2]^{n_2} \ldots Q_m \boldsymbol{x}_m \in [-c_m, c_m]^{n_m} \; \Phi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$$

is valid.

Thus, for a formula $\Phi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$, we can design a counter program that simulates the formula by using bounded counters for $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$. We do this in the next section by designing different components that can simulate the atomic predicates inside the formula and also the universal and existential quantifiers.

4.3.2 Components for arithmetic theories

To start, we observe that the variables of short Presburger arithmetic range over the integers, but counter programs work with non-negative counters. Thus, we use two counters to represent an integer-valued counter in the following way: for a values $v \in \mathbb{Z}$, we use non-negative counters v_+ and v_- such that $v = v_+ - v_-$. Also, we say that the representation of a variable v is *normalised* in a configuration (L, f) if and only if $f(v_+) = 0$ or $f(v_-) = 0$. If we want to be able to perform zero tests on this variable, we also introduce variables $\hat{v_+}, \hat{v_-}$ as described in Section 4.2.3.

For loops

The first component that we introduce is a new **loop** instruction, namely, **for** loops. This can be seen as syntactic sugar for a larger program and it will be a useful tool for future components. Let C be a counter program such that C does neither mention the counters v nor t. Here, t is a temporary variable that is assumed to be initialised with the value zero. Moreover, assume that C is deterministic with respect to valid runs, i.e., when run starting from a valid configuration, there is exactly one valid terminating run. We write

for v **do**
    C

as an abbreviation for

1: **loop**
2: $\quad v \mathrel{-}= 1; C; t \mathrel{+}= 1$
3: **zero-test**(v)
4: **loop**
5: $\quad v \mathrel{+}= 1; t \mathrel{-}= 1$
6: **zero-test**(t)

The desired effect is that C is executed v times and at the end the value of v is restored. The properties of this **for** loop are that if the initial configuration is valid and $\mathrm{val}(t) = 0$ in that configuration, then the following hold:

1. The values of the counters v and t are not changed by any valid run: The loop in Line 1 transfers the value of v to t, this is guaranteed by the zero test in Line 3. The loop in Line 4 together with the zero test in Line 6 then re-establish the value of v and ensure that t has value zero.

2. The effect of any valid run π is $val_1(v) \cdot \text{eff}(\pi)$: Any accepting run must pass the zero test on Line 3, so the loop on Line 1 is executed val(v) many times, and thus the effect of any accepting run is $val_1(v) \cdot \text{eff}(C)$.

A few comments are in order. If multiple **for** loops are sequentially composed, we can clearly re-use the temporary variable t. However, whenever we nest multiple **for** loops this is not possible, and we assume that we have n different temporary variables, where n is the maximum nesting depth of the **for** loops.

We can now describe components that implement primitive arithmetic operations and predicates.

Leq$[u, v] : u \leq v$**.**

We define a component that enables testing whether the value of some variable is less than the value of some other stored variable. Formally, we require the following properties if I holds in the initial configuration:

1. The effect of any valid run is zero (except for the testing counters);

2. LEQ$[u, v]$ admits a valid run if and only if $val_1(u) \leq val_1(v)$; and

3. LEQ$[u, v]$ is a component.

The component LEQ$[u, v]$ implementing those requirements is as follows:

```
1: goto 2 or 9 or 16
2: zero-test(u_−)
3: zero-test(v_−)                          ▷ Case 1: val(u), val(v) ≥ 0
4: for u_+ do
5:     v_+ −= 1                            ▷ Test if u_+ reaches zero before v_+
6: for u_+ do
7:     v_+ += 1                            ▷ Restore the value of v_+
8: goto 18
9: zero-test(u_+)
```

10: **zero-test**(v_+) ▷ Case 2: $val(u), val(v) \leq 0$
11: **for** v_- **do**
12: $\quad u_- \mathrel{-}= 1$ ▷ Test if v_- reaches zero before u_-
13: **for** v_- **do**
14: $\quad u_- \mathrel{+}= 1$ ▷ Restore the value of u_-
15: **goto** 18
16: **zero-test**(u_+)
17: **zero-test**(v_-) ▷ Case 3: $val(u) \leq 0$ and $val(v) \geq 0$
18: **skip**

Property 1. The effect of a valid run that branches to Line 2 is to decrease the value of v_+ by the value of u_+ and then increase it again by the value of u_+. Similarly, a valid run that branches to Line 9 decrease the value of u_- by the value of v_- and then increase it again by the value of v_-. Finally, a valid run that branches to Line 16 has no effect on the value of the non-testing counters.

Property 2. Assume that LEQ$[u, v]$ admits a valid run. Any valid run is a result of three different cases:

- Case 1: $val_1(u_-) = val_1(v_-) = 0$. We decrease the value of v_+ by the value of u_- and then restore the value of v_+. This can only happen if $val_1(v_+) \geq val_1(u_+)$ and since $val_1(u_-) = val_1(v_-) = 0$ we get that $val_1(u) \leq val_1(v)$.
- Case 2: $val_1(u_+) = val_1(v_+) = 0$. Analogously.
- Case 3: $val_1(u_+) = val_1(v_-) = 0$. Then $val_1(u) \leq 0$ and $val_1(v) \geq 0$, and thus we must have that $val_1(u) \leq val_1(v)$.

For the other direction, assume that $val_1(u) \leq val_1(v)$. By I we know that the counter values are normalised, so it must the case that $val_1(u_+) \leq val_1(v_+)$ and $val_1(u_-) = val_1(v_-) = 0$ or $val_1(v_-) \leq val_1(u_-)$ and $val_1(u_+) = val_1(v_+) = 0$ or $val_1(u_+) = val_1(v_-) = 0$. For each case there exist a valid run of LEQ$[u, v]$ that branches to Lines 2, 9 or 16 respectively.

Property 3. Notice that any valid run can perform at most three **zero** tests for u and v, and two **zero** tests for t. Property 1 implies that the invariant I is maintained by any valid run and the testing counters are updated only by **zero-test**() instructions.

Add$[r, v] : r \mathrel{+}= v$**.**

The next component $\textsc{Add}[r, v]$ that we define enables us to add the value stored in the counter v to the counter r. The properties we require are as follows:

1. After any valid run, all counter values are preserved apart from the testing counters and the counters associated to r;
2. The effect of any valid run that starts in a configuration where I holds is that once the run terminates, $\text{val}(r) = \text{val}_1(r) + \text{val}_1(v)$; and
3. $\textsc{Add}[r, v]$ is a component.

The component $\textsc{Add}[r, v]$ is now defined as follows:

1: **goto** 2 **or** 6
2: **zero-test**(v_-) ▷ Case 1: $v \geq 0$
3: **for** v_+ **do**
4: $\quad r_+ \mathrel{+}= 1$ ▷ Add $val(v_+)$ to r_+
5: **goto** 9
6: **zero-test**(v_+) ▷ Case 2: $v < 0$
7: **for** v_- **do**
8: $\quad r_- \mathrel{+}= 1$ ▷ Add $val(v_-)$ to r_-
9: **loop** ▷ Normalise r
10: $\quad r_+ \mathrel{-}= 1; r_- \mathrel{-}= 1$
11: **goto** 12 **or** 14
12: **zero-test**(r_+)
13: **goto** 15
14: **zero-test**(r_-)
15: **skip**

Property 1. The values of counters v_+, v_-, t are preserved by the properties of the **for** loop. Since, the only variables involved in $\textsc{Add}[r, v]$ are r, v, t, all other counter values are preserved by $\textsc{Add}[r, v]$.

Property 2. Consider a valid run π, we distinguish two cases:

- $val_1(v) \geq 0$: Then π must branch to Line 2 in order to pass the zero test. The loop on Line 3 increases the value of r_+ by $val_3(v_+)$, so $val_5(r) = val_1(r) + val_1(v)$ since $val_3(v_-) = 0$. The only change that π performs on the counters of r after Line 5 is on Line 10, but this operation does not change $val(r)$.

- $val_1(v) < 0$: Then π must branch to Line 6 in order to pass the zero test. Next, the loop on Line 7 increases the value of r_- by $val_7(v_-)$, so $val_9(r) = val_1(r) - val_7(v_-) = val_1(r) = val_1(r) + val_1(v)$, since $val_1(v_+) = 0$. Again, the only operation that π can perform on the counters of r after this step is on Line 10, but this operation does not change $val(r)$.

Property 3. It is clear that the values of the testing counters are only updated by **zero-test**() instructions. Also, notice that any valid run performs exactly two zero tests on the counters of v, one zero test on the counters of r, and one zero test on t. Finally, the fact that the variable v remains normalised is implied by Property 1. For the variable r, we observe that any accepting run must either pass the zero test on Line 12 or the one on Line 14, so r must be normalised as well. Also, the value of the counter t remains zero by Property 1.

Mul$[r, u, v] : r \mathrel{+}= u \cdot v$**.**

We now define a component $\textsc{Mul}(r, u, v)$ that enables adding the product of the values of the counters u and v to the counter r. To simplify the presentation, we first provide a component $\textsc{Sgn}[u] : u = -u$. The properties of $\textsc{Sgn}[u]$ are:

1. After a valid run, all counter values are preserved apart from the counters associated to the variable u;
2. When run in a configuration where I holds, we have that $val(u)$ after the component terminates is $-val_1(u)$; and
3. $\textsc{Sgn}[u]$ is a component.

The definition of $\textsc{Sgn}[u]$ is as follows:

1: **goto** 2 **or** 7
2: **zero-test**(u_-) ▷ Case 1: $val(u) \geq 0$
3: **loop**
4: $\quad u_+ \mathrel{-}= 1; u_- \mathrel{+}= 1$
5: **zero-test**(u_+)
6: **goto** 11
7: **zero-test**(u_+) ▷ Case 2: $val(u) \leq 0$
8: **loop**
9: $\quad u_+ \mathrel{+}= 1; u_- \mathrel{-}= 1$
10: **zero-test**(u_-)

11: **skip**

Property 1. Clearly, $\textsc{Sgn}[u]$ does not mention any counters apart from those associated to u.

Property 2. Assume $val_1(u) > 0$. Since u is normalised, it must be the case that $val_1(u_-) = 0$ and $val_1(u_+) \geq 0$, so any valid run must branch to Line 2 to pass the zero test. Line 4 decreases the value of u_+ by some non-deterministically chosen n and increases the value of u_- by the same n. To pass the zero test on Line 5, we must have that $n = \mathrm{val}_3(u_+) = \mathrm{val}_3(u)$ since $\mathrm{val}_3(u_-) = 0$ and thus $\mathrm{val}_{11}(u) = \mathrm{val}_3(u) - 2n = \mathrm{val}_3(u) - 2\,\mathrm{val}_3(u) = -\,\mathrm{val}_1(u)$. If $\mathrm{val}_1(u) = 0$, $\mathrm{val}_1(u_+) = \mathrm{val}_1(u_-) = 0$ then any valid run can jump to either Line 2 or 7, but it cannot make any changes to the values of the counters associated with u. Finally, if $\mathrm{val}_1(u) < 0$, any valid run must jump to Line 7 to pass the zero test. Line 9 increases the value of u_+ by some non-deterministic n and decreases the value of u_- by n. To pass the zero test on Line 5 we must have that $n = -\,\mathrm{val}_8(u_-) = -\,\mathrm{val}_8(u)$ since $\mathrm{val}_8(u_+) = 0$ and thus $\mathrm{val}_{11}(u) = \mathrm{val}_8(u) + 2n = \mathrm{val}_8(u) - 2\,\mathrm{val}_8(u) = -\,\mathrm{val}_1(u)$.

Property 3. $\textsc{Sgn}(u)$ is a component because any run can execute at most two zero tests on the variable u, and any valid run must either pass the zero test on Line 5 or the one on Line 10, so the effect of any accepting run cannot make variable u un-normalised.

We are now fully prepared to introduce the component $\textsc{Mul}[r, u, v]$. The properties we require of any valid run of $\textsc{Mul}[r, u, v]$ that starts in a configuration where I holds are as follows:

1. The component does not change the value of counters that do not involve variables u, v, r, t, and once the component terminates, we have $\mathrm{val}(u) = \mathrm{val}_1(u)$, $\mathrm{val}(v) = \mathrm{val}_1(v)$, and $\mathrm{val}(r) = \mathrm{val}_1(r) + \mathrm{val}_1(u) \cdot \mathrm{val}_1(v)$; and

2. $\textsc{Mul}[r, u, v]$ is a component.

The component $\textsc{Mul}(r, u, v)$ is now defined as follows:

1: **goto** 2 **or** 6
2: **zero-test**(u_-) ▷ Case 1: $val(u) \geq 0$
3: **for** u_+ **do**
4: $\quad$ $\textsc{Add}(r, v)$ ▷ $val(r)$ += $val(u) \cdot val(v)$
5: **goto** 11
6: **zero-test**(u_+) ▷ Case 2: $val(v) \leq 0$
7: $\textsc{Sgn}(v)$ ▷ $val(v) = -val(v)$

8: **for** u_- **do**
9: ADD(r, v) ▷ $val(r)$ += $val(u) \cdot val(v)$
10: SGN(v) ▷ Restore $val(v)$
11: **skip**

Property 1. It is clear that the effect of any valid run maintains the values of all variables apart from r, u, v, t since the counters used for these values are not mentioned in the counter program.

- Let π be a valid run that branches to Line 2. We have that $\text{val}_3(u_-) = \text{val}_1(u_-) = 0$ since π passes the zero test on Line 3. Also, after the loop on Line 3, by the properties of ADD, we have that $val_5(r) = \text{val}_1(r) + \text{val}_1(u_+) \cdot \text{val}_1(v)$. Since $\text{val}_1(u_-) = 0$, we conclude that $\text{val}_5(r) = \text{val}_1(r) + \text{val}_1(u) \cdot \text{val}_1(v)$, so $\text{val}(r) = val_1(r) + val_1(u) \cdot val_1(v)$ once the component terminates. Finally, because **for** and ADD maintain the values of variables u and v, we can conclude that $val(v) = val_1(v)$ and $val(u) = val_1(u)$ once the component terminates.

- Now, assume π branches to Line 6. Since π passes the zero test on Line 6, we have that $\text{val}_1(u_+) = 0$. By the properties of SGN we get that $\text{val}_8(v) = -\text{val}_1(v)$. Next, by the properties of **for** and ADD we get that

$$\begin{aligned} \text{val}_{10}(r) &= \text{val}_8(r) + \text{val}_8(u_-) \cdot \text{val}_8(v) \\ \text{val}_{10}(v) &= \text{val}_8(v) \\ \text{val}_{10}(u_-) &= \text{val}_8(u_-)\,. \end{aligned}$$

  Replacing $\text{val}_8(v)$ with $-\text{val}_1(v)$ gives us

$$\begin{aligned} \text{val}_{10}(r) &= \text{val}_1(r) - \text{val}_1(u_-) \cdot \text{val}_1(v) \\ \text{val}_{10}(v) &= -\text{val}_1(v)\,. \end{aligned}$$

  Because π does not change the value of u_+ or u_-, we can conclude that $\text{val}(u) = val_1(u)$ once the component terminates. Moreover, $\text{val}_{10}(v) = -\text{val}_1(v)$, so $\text{val}(v) = \text{val}_1(v)$. Finally, since $\text{val}_1(u_+) = 0$, we have that $\text{val}_1(u) = -\text{val}_1(u_-)$, so $\text{val}_{end}(r) = \text{val}_1(r) + \text{val}_1(u) \cdot \text{val}_1(v)$.

Property 2. It is clear that any valid run of MUL$[r, u, v]$ updates the testing counters only by **zero-test**() instructions, and the number of such zero tests performed is bounded by $2 \cdot u_{max}$ for the variables r, v, and bounded by two for the variables u and t. Moreover, the only changes to the counters associated to u, v and

r are performed inside the components ADD, SGN and FOR, so the variables remain normalised. Thus, MUL$[r, u, v]$ is a component.

Having defined this last component, we are able to model atomic formulas of short Presburger arithmetic. We now need to describe how we deal with Boolean combinations of such predicates, and how to model quantification. Before we do this, we make some remarks related to Rudimentary arithmetic. We can reuse most of the components to model the predicates of Rudimentary arithmetic, with two small changes. First, things are a a bit simpler since we do not have to deal with negative integers. Secondly, we allow for multiplication between variables, i.e. we can write formulas such as $x^2 + xy + y^2 = 1$. Before, our multiplication component dealt with the multiplications of the type $a \cdot x$, where a is a constant and x a variable and was only preserving the result of $a \cdot x$ inside the variable x. Now, we want to preserve both values and use a third variable to store the result, thus we introduce the following component.

Copy.

We provide a counter program COPY$[x, x']$ with the following properties:

1. it admits a valid run if and only if $\mathrm{val}_{\mathrm{end}}(x') = \mathrm{val}_{\mathrm{end}}(x) = \mathrm{val}_1(x)$; and
2. COPY$[x, x']$ is a component.

We implement COPY$[x, x']$ as follows:

1: **loop**
2: $\quad x' -= 1$
3: **zero-test**(x')
4: **loop**
5: $\quad x -= 1; x' += 1; t += 1$
6: **zero-test**(x)
7: **loop**
8: $\quad t -= 1; x += 1$
9: **zero-test**(t)

The loop on line 1 ensures that $\mathrm{val}_4(x') = 0$. We do not do this for the auxiliary counter t because any valid run sets $\mathrm{val}_{\mathrm{end}}(t) = 0$. Observe that COPY$[x, x']$ admits a valid run if and only if the loop on line 4 is executed $\mathrm{val}_1(x)$ many times and the loop on line 7 is executed $\mathrm{val}_4(t) = \mathrm{val}_1(x)$ many times which happens if and only if $\mathrm{val}_{\mathrm{end}}(x') = \mathrm{val}_{\mathrm{end}}(x) = \mathrm{val}_1(x)$. Moreover, any valid run performs 3 calls to

the **zero-test**() subroutine, so COPY$[x, x']$ is a component. The auxiliary counter t is reused between different COPY components, so that we do not introduce new counters every time we use the COPY component. Notice that the programs become a lot simpler when we do not need to consider negative integers. As an example, the component that models addition for Rudimentary arithmetic is the following. The counter program ADDITION$[x, y, z]$ enables us to check whether the value stored in counter z is equal to the sum of the values stored in x, y. Formally, it has following properties:

1. ADDITION$[x, y, z]$ admits a valid run if and only if $\text{val}_1(x) + \text{val}_1(y) = \text{val}_1(z)$;
2. ADDITION$[x, y, z]$ is a component; and
3. the effect of ADDITION$[x, y, z]$ is zero on counters x, y, z.

We implement ADDITION$[x, y, z]$ as follows:

1: COPY$[x, x']$; COPY$[y, y']$; COPY$[z, z']$
2: **loop**
3: $\quad z' -= 1$
4: $\quad x' -= 1$ **or** $y' -= 1$
5: **zero-test**(x'); **zero-test**(y'); **zero-test**(z')

It is easy to see that the first property is fulfilled by the counter program and that ADDITION$[x, y, z]$ is a component because any run performs exactly 12 calls to **zero-test**() (9 calls on line 1, and 3 calls on line 5). The last property is true based on the properties of COPY. The component for the negation of the addition predicate can be defined similarly.

4.3.3 Components for quantification

We now define components that allow us to existentially and universally quantify over variables of a bounded range.

Exists$[v]$**.**

We begin with the existential quantifier and design a new component EXISTS$[v]$. The properties of EXISTS$[v]$ we require are as expected:

1. for every $u \in [-v_{max}, v_{max}]$, there exists a valid run π such that $\text{val}(v) = u$ in the terminating configuration of π; and

2. Exists$[v]$ is a component.

We define Exists$[v]$ as the following counter program:

1: **loop**
2: $\quad v_+ \mathrel{-}= 1$
3: **loop**
4: $\quad v_- \mathrel{-}= 1$
▷ Non-deterministically decrement v_+ and v_-
5: **loop**
6: $\quad v_+ \mathrel{+}= 1$
7: **loop**
8: $\quad v_- \mathrel{+}= 1$
▷ Non-deterministically increment v_+ and v_-
9: **goto** 10 **or** 12
10: **zero-test**(v_+)
11: **goto** 13
12: **zero-test**(v_-) ▷ Ensure v is normalised
13: **skip**

Property 1. Assume $u \geq 0$. Then there exists a run π that traverses the first two loops to set v_+ and v_- to zero, uses the third loop to set v_+ to u and does not enter the fourth loop. Analogously, for $u < 0$ there exists run π that uses the first two loops to set v_+ and v_- to zero, does not traverse the third loop and traverses the fourth loop to set v_- to $|u|$.

Property 2. Clearly, only one zero test is performed by any run on the variable v. Also, assuming $I(v)$ holds initially, this is still the case when the component terminates since a valid run passes either the zero test on Line 10 or the one on Line 12. Recall that the instruction $v_+ \mathrel{-}= 1$ is syntactic sugar for $v_+ \mathrel{-}= 1; \hat{v_+} \mathrel{+}= 1$, so $\mathrm{val}(v) \in [-v_{max}, v_{max}]$ once Exists$[v]$ terminates.

ForAll$[v] : C[v]$**.**

The construction for a component simulating a universal quantifier is more involved. Let $C[v]$ be a component that may access the counter v, test it for zero and change its value on intermediate steps, but has overall net effect zero on counter v. We write ForAll$[v] : C[v]$ for the following counter program:

1: **loop**
2: $\quad v_- \mathrel{+}= 1$

3: **loop**
4: $\quad v_+ \mathrel{-}= 1$
5: **zero-test**(v_+)
6: **zero-test**$(\widehat{v_-})$ ▷ Initialise the value of v to $-v_{max}$
7: $C[v]$
8: $v_- \mathrel{-}= 1$
9: **goto** 7 **or** 10
10: **zero-test**(v_-)
11: $C[v]$
12: $v_+ \mathrel{+}= 1$
13: **goto** 11 **or** 14
14: **zero-test**$(\widehat{v_+})$
15: $C[v]$
16: **skip**

The properties of ForAll$[v] : C[v]$ are as follows:

1. there is a valid run if and only if for all $u \in [-v_{max}, v_{max}]$, C has a valid run with $\mathrm{val}_1(v) = u$
2. ForAll$[v] : C[v]$ is a component.

Property 1. Let π be a valid run. The first time π reaches Line 7, we have $\mathrm{val}_7(\widehat{v_-}) = 0$ and hence $\mathrm{val}_7(v_-) = v_{max}$. Since a valid run of $C[v]$ does not change the value of v when it terminates, we have that π on Line 9 jumps to Line 7 v_{max} many times. Subsequently, π jumps to Line 10 in order to pass the zero test there. At this point we know that $val_{11}(v_-) = 0$ and that π executed three zero tests on v and $C[v]$ was executed v_{max} many times. Also, once π reaches Line 10, it must be the case that for all $u \in [-v_{max}, -1]$, $C[v]$ admits a valid run with $\mathrm{val}_1(v) = u$.

The first time π reaches Line 10, we have that $\mathrm{val}_{10}(v_+) = 0$ because π passes the zero test on Line 5 and it cannot change the value of v_+ before the first time it reaches Line 10. So, in order for π to pass the zero test on Line 14, it must execute the instructions $11 \to 12 \to 13$ v_{max} many times. Thus, once π reaches Line 15, we know that for all $u \in [-v_{max}, v_{max} - 1]$, $C[u]$ admits a valid run and that π executes exactly 4 zero tests on v in order to reach Line 15 and executes $C[v]$ $2v_{max}$ many times. Finally, for π to be a valid run, we require that C has a valid run with $\mathrm{val}_1(v) = v_{max}$.

Putting everything together, if the program has a valid run then for all $u \in [-v_{max}, v_{max}]$, C admits a valid run with $\mathrm{val}_1(v) = u$. Any valid run maintains I

because it executes exactly four zero tests on v and executes $C[v]$ exactly $2v_{max}+1$ many times.

Property 2. If $C[v]$ is a component, the construct ForAll$[v]; C[v]$ is a component since the number of zero tests performed is bounded by $2 + 2v_{max} \cdot b$, where b is maximum number of zero tests performed by $C[v]$. Finally, one can see that the values of the testing counters of v are changed only with the zero test instructions. This is the first time we use a zero test for counters $\widehat{v_-}, \widehat{v_+}$, but there is nothing special about this and we do not break the invariant by doing this. Also, since $C[v]$ is a component, we know that the values of the testing counters are only updated by **zero-test**() instructions. The last thing that we need to argue is that if v is normalised in the initial configuration, then v is normalised at the end. Notice that any accepting run must pass the zero test on Line 10 and after that the run cannot change the value of v_-. So after any accepting run, $val_{end}(v_-) = 0$, so v is normalised.

Again, these two components can be also used to model the existential and universal quantifiers of Rudimentary arithmetic. In this case however, we also need to have a counting quantifier. We design a component which is an extension of the ForAll$[v] : C[v]$ component. Formally, ExistsC$[x, v] : C[v]$ component has the following properties:

- it admits a valid run if and only if there exist more than $\text{val}_1(x)$ different integers $n \in \underline{N}$ such that C has a valid run with $\text{val}_1(v) = n$
- the overall effect on counter x is zero; and
- ExistsC$[x, v] : C[v]$ is a component.

We write ExistsC$[x, v] : C[v]$ for the following counter program:

1: **loop**
2: $\quad v -= 1$
3: **zero-test**(v)
4: Copy$[x, x']$
5: **goto** 6 **or** 9
6: **zero-test**$(\hat{x'})$
7: ForAll$[v] : C[v]$
8: **goto** 15
9: $x += 1$
10: **loop**
11: $\quad v += 1$

12: **goto** 13 **or** 10

13: $C[v]$; $x' -= 1$

14: **zero-test**(x')

15: **halt**

Observe that the counter program is very similar to the counter program $\textsc{ForAll}[v] : C[v]$. The branching on line 5 checks whether $\mathrm{val}_1(x) = N$. If so, $C[v]$ must have a valid run for all values of v, which is checked on line 7. Otherwise, the instructions on line 13 ensure that the value of x' can be decremented if only if $C[v]$ admits at least one valid run with the current value of v. Moreover, the zero test on line 14 is passed if and only if $C[v]$ admitted a valid run for more than $\mathrm{val}_1(x)$ different values. Similarly to the $\textsc{ForAll}$ case, since $C[v]$ is a component, we have that it makes at most a polynomial number of calls to **zero-test**(). If we denote this number by B, the maximum number of calls to **zero-test**() performed by $\textsc{ExistsC}[x, v] : C[v]$ is bounded by $N \cdot B + 5$. Hence, it is indeed a component.

4.3.4 Putting it all together

Having defined all the building blocks above, we are now able to prove the following proposition.

Proposition 29. *For any formula Ψ of Short-PA$_m$, there exists a component C and an initial configuration $(0, f)$ whose encoding is polynomial in $|\Psi|$ such that Ψ is valid if and only if C has a valid run starting in $(0, f)$.*

Proof. Let $\Psi \equiv \exists \boldsymbol{x}_1 \forall \boldsymbol{x}_2 \cdots Q_m \boldsymbol{x}_m \, \Phi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$ be a sentence of Short-PA$_m$ and let $\boldsymbol{x}$ be the vector of all K variables in Ψ, and $\boldsymbol{a}$ the vector of all constants occurring in Ψ. The matrix Φ is a positive Boolean combination of atomic formulas of the form $\boldsymbol{a}' \cdot \boldsymbol{x}' \leq b$ with all values in $\boldsymbol{a}', b$ occurring in $\boldsymbol{a}$ and all variables of $\boldsymbol{x}'$ occur in $\boldsymbol{x}$. Moreover, let $p_1, \ldots, p_m \colon \mathbb{N} \to \mathbb{N}$ be the polynomials obtained from Proposition 28.

The counters of the component C are defined to be:

- a counter in vector $\boldsymbol{x}_C$ corresponding to every variable in Ψ;
- a counter in vector $\boldsymbol{a}_C$ corresponding to every constant in Ψ; and
- the relevant auxiliary counter variables used in Section 4.3.2, in particular r used in the component $\textsc{Mul}(r, u, v)$.

We initialise them inside $(0, f)$ as follows:

- $f(x_C) = 0$ and $f(\hat{x_C}) = 2^{p_i(|\Psi|)}$ for all variables x_C occurring in $\boldsymbol{x}_C$;
- $f(a_C) = a$ and $f(\hat{a_C}) = 0$ for all variables a_C occurring in $\boldsymbol{a}_C$;
- $f(r) = 0$ and $f(\hat{r}) = 2^{p_m(|\Psi|)} \cdot ||\boldsymbol{a}||_1$; and
- for the testing counters, we pick as maximum value $(2||\boldsymbol{x}||_\infty)^{(mK+1)} \cdot (||\boldsymbol{a}||_\infty + 1)$.

Note that $(0, f)$ can indeed be computed in polynomial time, assuming binary encoding of numbers.

For an atomic proposition $\boldsymbol{a}' \cdot \boldsymbol{x}' \leq b$, we can construct a fixed component $\textsc{Leq}(\boldsymbol{a}', \boldsymbol{x}', b, r)$ such that the inequality is true if and only if $\textsc{Leq}(\boldsymbol{a}', \boldsymbol{x}', b, r)$ admits a valid run starting in (L, f). Thus, we can construct a component C_Φ that admits a valid run starting from some valid configuration $(0, f)$ if and only if Φ evaluates to true under the assignment induced by $(0, f)$, by simulating a conjunction via sequential composition and disjunction by non-deterministic branching. Notice that in particular the counter r used to evaluate $\boldsymbol{a}' \cdot \boldsymbol{x}' \leq b$ has $\hat{r}$ large enough to compute $\boldsymbol{a}' \cdot \boldsymbol{x}'$ for all values of $\boldsymbol{x}'$ attained by the relativised quantifiers.

We now turn towards simulating the quantifiers in Ψ. For $1 \leq i \leq m$ and $\boldsymbol{x}_i = (x_1, \ldots, x_{m_i})$, the component C_i is obtained as follows:

- If $Q_i = \exists$ then C_i is the component
  1: $\textsc{Exists}[x_1]$
  2: $\vdots$
  3: $\textsc{Exists}[x_{m_i}]$
  4: C_{i+1}
- If $Q_i = \forall$ then C_i is the component
  1: $\textsc{ForAll}[x_1]$:
  2: $\vdots$
  3: $\textsc{ForAll}[x_{m_i}] : C_{i+1}$

Finally, we let $C = C_1$ and $C_{m+1} = C_\Phi$. By the properties established in Section 4.3.2, it is clear that C admits a valid run starting in $(0, f)$ defined above if and only if Ψ is valid. Indeed, the sequential composition of the components C_1 to C_m correctly implements the range-bounded alternating quantification required in Proposition 28. Every time C_Φ is executed, due to C_Φ being a component it is guaranteed that it is traversed by a valid run if and only if Φ evaluates to true under the assignment encoded in the configuration when entering C_Φ.

For valid runs to exist, it remains to be shown that any valid run of C_1 cannot perform more than $(2||\boldsymbol{x}||_\infty)^{(mK+1)} \cdot (||\boldsymbol{a}||_\infty + 1)$ zero tests. By the properties of $\textsc{Leq}$, we know that any valid run in C_Φ performs at most $2(||\boldsymbol{a}||_\infty + 1)$ zero tests on any variable. Furthermore, if any valid run of C_{i+1} performs at most B zero tests for any variable then any valid run in C_i performs at most the following number of zero tests:

- $B + K \leq 2B$ if $Q_i = \exists$; and
- $(2||\boldsymbol{x}||_\infty)^{p_m(|\Psi|)}B$ if $Q_i = \forall$.

Thus, any valid run of C performs at most $(2x_M)^{mN+1}(a_M + 1)$ zero tests on any variable.

Finally, observe that C itself is fixed by the structure of Ψ. In particular, the only variable part, the coefficients $\boldsymbol{a}$, are fully encoded in the initial configuration $(0, f)$. □

We are now ready to introduce the first complexity result of this chapter. Recall the problem of Counter program zero-reachability and consider the setting in which the counter program is fixed.

Problem 30. FIXED COUNTER PROGRAM ZERO-REACHABILITY
Input: *A vector $\boldsymbol{x} \in \mathbb{N}^d$ of initial values of the counters.*
Fixed: *A counter program $\mathcal{C}$.*
Output: *YES if and only $\mathcal{C}$ has a zero-terminating run from $\boldsymbol{x}$.*

Using the above proposition, we get the following theorem.

Theorem 31. *For any $i > 0$, there is a fixed counter program $\mathcal{C}_i$ such that deciding whether $\mathcal{C}_i$ has a zero-terminating run is Σ_i^{P}-hard.*

An instance of an acceptance problem for a fixed counter program directly corresponds to a reachability query in a VASS. To prove the above theorem, it remains to observe that the question of deciding whether C in Proposition 29 has a valid run reduces to an instance of an acceptance problem by non-deterministically decrementing all counters once C has terminated, and halting if all counters are equal to zero. This concludes the proof of Theorem 31. In Section 4.5 we upgrade the result of Theorem 31 by providing a PSPACE lower bound for the fixed VASS zero-reachability problem, however, we believe that the connection to Short-PAis relevant on its own and it also paves the way for the results in the next section.

By the use of similar arguments, one can obtain an almost identical version of Proposition 29 for rudimentary arithmetic. Before presenting the result, we first need a discussion on unary problems.

4.4 Hardness for unary counters

In order to study decision problems whose input is, for some constant k, a k-tuple of numbers presented in unary, and hence to analyse languages corresponding to them, we need a notion of reductions that are weaker compared to the standard ones that are widely used in computational complexity. The reason is that classical problems involving numbers represented in unary, such as Unary Subset Sum [38], have as an input a variable-length sequence of numbers given in unary. Hence, languages of such problems are in fact binary, as we need a delimiter symbol to separate the elements of the sequence. It is not clear how a reasonable reduction from such a language to a language consisting of k-tuples of numbers for a *fixed* k would look like. In particular, note that unary Fixed VASS zero-reachability is not the unary "counterpart" of binary Fixed VASS zero-reachability in the classical sense. Conversely, arithmetic properties of a single number, e.g. primality or square-freeness, require very low computational resources if the input is represented in unary. Hence, the notion of a reduction between such "genuinely unary" languages has to be very weak.

In view of this discussion, we introduce the following kind of reduction. Given $k > 0$, a *k-tuple unary language* is a subset $L \subseteq \mathbb{N}^k$. We say that L is a tuple unary language if L is a k-tuple unary language for some $k > 0$. Let $L \subseteq \mathbb{N}^k$ and $M \subseteq \mathbb{N}^\ell$ be tuple unary languages, we say that L *arithmetically reduces* to M if there are fixed polynomials $p_1, \ldots, p_\ell \colon \mathbb{N}^k \to \mathbb{N}$ such that $(m_1, \ldots, m_k) \in L$ if and only if $(p_1(m_1, \ldots, m_k), \ldots, p_\ell(m_1, \ldots, m_k)) \in M$.

We believe that this reduction is sensible for the following informal reasons. Polynomials can be represented as arithmetic circuits. To the best of our knowledge, there are no known lower bounds for, e.g. comparing the output of two arithmetic circuits with all input gates having value one [1], suggesting that evaluating a polynomial is a computationally weak operation. Moreover, in the light of sets of numbers definable in rudimentary arithmetic, it seems implausible that applying a polynomial transformation makes, e.g. deciding primality of a number substantially easier.

For a formula Φ, let $\mathcal{L}_\Phi$ be the tuple unary language of yes-instances for Fixed rudimentary $\mathbf{FO}k\text{-}\mathbf{aryC}(+,\times)$ validity. Also, for a counter program C, define $\mathcal{L}_C$ as the tuple unary language of yes-instance for the Fixed counter program zero-reachability problem. Now we are ready to introduce the following theorem, which is a consequence of a modified version of Proposition 29.

Theorem 32. *For every formula* Φ *of rudimentary* **FO**k**-aryC**$(+,\times)$*, there exists a counter program* C *such that* $\mathcal{L}_\Phi$ *arithmetically reduces to* $\mathcal{L}_C$*.*

This theorem can be viewed in two different contexts. On the one hand, it relates the computational complexity of the two problems using a very weak reduction as described above. On the other hand, it also relates the expressivity of two formalisms. Namely, the set of satisfying assignments for formulas of rudimentary arithmetic is at most as expressive as the composition of polynomial transformations with the sets of initial configurations for zero-reachable runs in counter programs. In particular, it shows that fixed VASS can, up to a polynomial transformation, decide number-theoretic properties such as primality, square-freeness, see [39] for further examples.

Now, we present the modified version of Proposition 29.

Proposition 33. *For any formula* $\Phi(\boldsymbol{x})$ *of* **FO**k**-aryC**$(+,\times)$*, there exists a component* C *over* k *counters and polynomials* $p_1,\dots,p_k : \mathbb{N}\times\mathbb{N}^n \to \mathbb{N}$ *such that for any* $N\in\mathbb{N}$ *and* $\boldsymbol{x}\in\mathbb{N}^n$*,* $\langle \underline{N},+,\times\rangle \models \Phi(\boldsymbol{x})$ *if and only if* C *admits a valid run from the initial configuration* $(p_1(N,\boldsymbol{x}),\dots,p_k(N,\boldsymbol{x}))$*.*

The proof of this proposition is very similar to the on of Proposition 29.

Proof. We prove this statement by structural induction on sub-formulas of Φ. As shown in [10], rudimentary **FOunC**$(+,\times)$ has the same expressive power as rudimentary **FO**k**-aryC**$(+,\times)$. Since in our setting the formula is fixed, we can thus assume that $\Phi\in$ **FOunC**$(+,\times)$. Moreover, it is easy to see that we can assume that only $\exists^{>x}$ is used as a counter quantifier, since $\exists^{=x}$ can easily be defined using it as described above. Finally, we can assume that negations appear in Φ only in front of arithmetic predicates. In particular, $\neg\exists^{>x}y\ \varphi(y)$ is equivalent to $(\exists^{>x'}y\ \neg\varphi(y))\wedge(x+x'=N)$.

The counters of the component C are defined to be:

- a counter in vector $\boldsymbol{x}_C$ corresponding to every free variable of $\Phi(\boldsymbol{x})$;
- a counter in vector $\boldsymbol{y}_C$ corresponding to every quantified variable of $\Phi(\boldsymbol{x})$;
- a counter in vector $\boldsymbol{a}_C$ corresponding to every constant of $\Phi(\boldsymbol{x})$; and
- the auxiliary counters $t_C,\boldsymbol{x}'_C,\boldsymbol{y}'_C,\boldsymbol{c}'_C$ used inside the components for predicates and counting quantifiers described above.

We initialise them as follows:

- $f_1(x_C) = x$ and $f_1(\hat{x}_C) = N - x$ for each counter x_C corresponding to a variable x in $\boldsymbol{x}$;
- $f_1(v) = 0$ and $f_1(\hat{v}) = N$ for all the counters corresponding to quantified variables and constants, and auxiliary counters; and
- for the testing counters, $f_1(u_1) = 2N$ and $f_1(u_2) = 2N \cdot P(N)$, where the polynomial $P(N)$ will be defined later.

Assume first that a sub-formula φ of Φ consists of a single literal. Then, by using the previously defined components, we can construct a fixed component C' corresponding to this literal. In C', for every valid initial configuration (L, f), there exists a valid run starting in it if and only if φ is true under the assignment of the values of the counters in (L, f) to the corresponding variables in φ. If φ is a Boolean combination of multiple literals, by simulating conjunction via sequential composition and disjunction by non-deterministic branching, we can construct a component C_φ with the same property.

Assume with no loss of generality that $\Phi(\boldsymbol{x})$ is in Prenex form, i.e.

$$\Phi(\boldsymbol{x}) \equiv \exists \boldsymbol{y}_1 \forall \boldsymbol{y}_2 \ldots Q_m \boldsymbol{y}_m \varphi(\boldsymbol{x}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_m)$$

and that the negations are pushed to the predicate level. Let M be the number of quantified variables, that is, the sum of the dimensions of $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m$, and K be the number of literals in $\Phi(\boldsymbol{x})$.

We now need to show how to simulate the quantifiers. Let C be the component constructed for φ. We then take

- for $\exists y\ \varphi$:
  1: Exists$[y_C]$
  2: $C[y_C]$
- for $\forall y\ \varphi$:
  1: ForAll$[y_C]$:
  2: $C[y_C]$
- for $\exists^{>x} y\ \varphi$:
  1: ExistsC$[x_C, y_C]$:
  2: $C[y_C]$

As noted above, to be able to use these components, we need to make sure that $C[y_C]$ has overall zero effect on the value of y_C. This is indeed true, since the only place where the value of a counter y_C is changed by a subroutine is in the component corresponding to the quantifier bounding y.

The counter program C starts with a component C_0 that initialises the counters $\boldsymbol{a}$ corresponding to the constants of $\Phi(\boldsymbol{x})$ by a sequence of instruction of the type $a \mathrel{+}= c$ for a corresponding constant c appearing in $\Phi(\boldsymbol{x})$. Finally, we let $C = C_0; C_1$. Indeed, the sequential composition of the components $C_1, \ldots, C_m$ correctly implements the range-bounded alternating quantification. Every time C_φ is executed, due to C_φ being a component it is guaranteed that it is traversed by a valid run if and only if

$\varphi(\boldsymbol{x}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_m)$ evaluates to true under the assignment encoded in the configuration when entering C_φ.

By the properties established in Section 4.3.2, it is clear that C admits a valid run starting with f_1 defined above if and only if $\Phi(\boldsymbol{x})$ is valid. To see that C is a component, it remains to note that at every step of the structural induction the number of calls to **zero-test**() is polynomial in N. Hence, there exists a polynomial $P(N)$ such that the overall number calls to **zero-test**() performed by C is bounded by $P(N)$. We conclude by reminding that we use this polynomial to initialise the value of the testing counter u_2.

We also show that any valid run of C cannot perform more than $(K \cdot (2N+12))^M$ zero tests. By the properties of our components, we know that any valid run in C_φ performs at most $K \cdot (2N + 12)$ zero tests on any counter. Furthermore, if any valid run of C_{i+1} performs at most B zero tests for any counter, then any valid run in C_i performs at most the following number of zero tests:

- $B + 1$ if $Q_i = \exists$; and
- $N^{m_i}B + 1$ if $Q_i = \forall$.

Thus, any valid run of C performs at most $(K \cdot (N + 12))^M$ zero tests in total on all counters. Finally, observe that C itself is fixed by the structure of $\Phi(\boldsymbol{x})$. In particular, the only variable part, the values of the free variables $\boldsymbol{x}$ and the bound N, are encoded only in the initial configuration f_1. □

4.5 A PSPACE lower bound for fixed VASS zero-reachability

The goal of this section is to show that there is a fixed 5-VASS whose zero-reachability problem is PSPACE-hard, provided that the initial configuration is encoded in binary.

We proceed with our construction as follows. We start with the halting problem for Turing machines (TMs) working in polynomial space and show that this problem is PSPACE-hard even if the space complexity of the TM is bounded by the length of its encoding and its input is empty. In Proposition 35, we then reformulate the halting problem as follows: given the encoding of such a machine as an input to a universal one-tape TM $\mathcal{U}$, does $\mathcal{U}$ accept?

We then use two consecutive simulations. First, we simulate $\mathcal{U}$ with a 3-counter automaton $\mathcal{A}$ (Proposition 36), and then simulate $\mathcal{A}$ with an 5-VASS $\mathcal{V}$ (Theorem 38).

To be able to apply the technique described in Section 4.2.3, we make sure that the space complexity stays linear in the size of the input throughout these simulations. This implies that both the upper bound on the value of the counters and the required number of zero tests are polynomial in the size of the input, which enables us to establish a polynomial time reduction. As a result we obtain a VASS $\mathcal{V}$ which, in a certain sense, can simulate arbitrary polynomial-space computations.

To provide the reduction, we then show how to transform in polynomial time the input of the problem we started with, the halting problem for polynomial-space TMs, into a zero-reachability query for $\mathcal{V}$.

4.5.1 The halting problem for space-bounded TMs

The goal of this subsection is to show that there exists a *fixed* polynomial-space TM whose halting problem is PSPACE-complete. Note that using standard arguments, we can assume that $\mathcal{M}$ below always halts.

Proposition 34. *[8, Section 4.2] The following problem is PSPACE-complete: given a TM $\mathcal{M}$, an input word w and a number n encoded in unary, decide if $\mathcal{M}$ accepts w in at most n space.*

We fix some way of encoding, using an alphabet of size at least two, of Turing machines and we denote by $\|\mathcal{M}\|$ the length of the encoding of $\mathcal{M}$, which we call the *size* of $\mathcal{M}$. Given a TM $\mathcal{M}$, we say that it is $\|\mathcal{M}\|$-space-bounded if on every input it halts using at most $\|\mathcal{M}\|$ space. Given $\mathcal{M}$, an input word w and a number n encoded in unary, it is easy to construct a $\|\mathcal{M}\|$-space-bounded TM $\mathcal{M}'$ such that if $\mathcal{M}$ accepts w in space at most n, then $\mathcal{M}'$ accepts on the empty input, otherwise $\mathcal{M}'$ rejects on the empty input. Moreover, the size of $\mathcal{M}'$ is polynomial in $\|\mathcal{M}\|$, $|w|$ and 2^n.

Indeed, $\mathcal{M}'$ can be constructed as follows. When run on the empty input, it writes w on some tape, and then runs $\mathcal{M}$ treating this tape as the input tape. Additionally, it initialises another tape with n written in unary, and before each step of $\mathcal{M}$ it checks that the space used by the tape where $\mathcal{M}$ is simulated does not exceed n. If it does, it immediately rejects. It is easy to see that such a TM is $\|\mathcal{M}'\|$-space-bounded and satisfies the required conditions.

Hence we get that the following problem is PSPACE-complete: given a $\|\mathcal{M}\|$-space-bounded TM $\mathcal{M}$, does $\mathcal{M}$ accept on the empty input? Observe that from the construction above we can assume that $\mathcal{M}$ has a special representation such that the fact that it is $\|\mathcal{M}\|$-space-bounded can be checked in polynomial time.

Let $\mathcal{U}$ be a one-tape universal TM. This TM has a single read-write tape, which in the beginning contains the input, that is, a description of a TM $\mathcal{M}$ it is going to simulate. If $\mathcal{M}$ is $\|\mathcal{M}\|$-space-bounded (and represented as mentioned in the previous paragraph), $\mathcal{U}$ simulates $\mathcal{M}$ on the empty input in space linear in $\|\mathcal{M}\|$ [8, Claim 1.6], otherwise $\mathcal{U}$ rejects. That is, in this space, $\mathcal{U}$ accepts or rejects depending on whether $\mathcal{M}$ accepts or rejects the empty word. Hence we get the following proposition.

Proposition 35. *There exists a fixed linear-space TM $\mathcal{U}$ such that the question whether $\mathcal{U}$ halts on a given input is PSPACE-complete.*

4.5.2 From TMs to a counter automata

In the previous subsection, we obtained a PSPACE-complete problem which already resembles the form of the reachability problem for a fixed counter program: given a fixed linear-space TM $\mathcal{U}$, does it accept a given input? In this section we show how to simulate $\mathcal{U}$ with a fixed counter automaton $\mathcal{A}$, and in the next section we show how to simulate $\mathcal{A}$ with a fixed binary VASS $\mathcal{V}$.

Let $\mathcal{A}$ be a counter automaton. We say that $\mathcal{A}$ is *deterministic* if for every configuration $(q, n_1, \ldots, n_d)$ there is at most one transition that $\mathcal{A}$ can take from this configuration. Suppose that $\mathcal{A}$ is deterministic, and that its final state q_f does not have any outgoing transitions. Let $\boldsymbol{n} = (n_1, \ldots, n_d) \in \mathbb{N}^d$. We treat $\mathcal{A}$ as an acceptor for such vectors. We say that $\mathcal{A}$ works in time t and space s on $\boldsymbol{n}$ if the unique run starting in the configuration $(q_0, n_1, \ldots, n_d)$ ends in a state without outgoing transitions, has length t, and the bit length of the largest value of a counter along this run is s. If this run ends in q_f, we say that $\mathcal{A}$ *accepts* this vector, otherwise we say that it *rejects* it. In all our constructions we make sure that there are no infinite runs. Note that, as in the case of TMs, we measure space complexity in the bit length of the values of the counters, and not in their actual values.

Let Σ be a finite alphabet. Let us bijectively assign a natural number to each word over Σ as follows. First, assign a natural number between 1 and $|\Sigma|$ to each symbol in Σ. Then w can be considered as a number in base $|\Sigma|+1$, with the least significant digit corresponding to the first letter of w. We denote this number by $\mathrm{num}(w)$.

Let $\mathcal{M}$ be a TM, and w be its input. We can transform the input w into a vector $(\mathrm{num}(w), 0, \ldots, 0)$, which will be the input of a deterministic counter automaton $\mathcal{A}$. We say that $\mathcal{A}$ *simulates* $\mathcal{M}$ if w is accepted by $\mathcal{M}$ if and only if the corresponding vector is accepted by $\mathcal{A}$. We say that this simulation is *in linear space* if there exists

a constant c such that if the space complexity of $\mathcal{M}$ is s on some input, then the space complexity of $\mathcal{A}$ on the coresponding input is cs.

The proof of the following proposition uses the techniques described in the proofs of [41, Theorem 4.3(a)] and [44, Theorem 2.4].

Proposition 36. *For every one-tape TM $\mathcal{M}$, there exists a deterministic 3-counter automaton $\mathcal{A}$ that simulates it in linear space.*

Proof. The idea of the proof is as follows. Two counters of $\mathcal{A}$, call them ℓ and r, represent the content of the tape of $\mathcal{M}$ to the left and to the right of the reading head. They are encoded similarly to the way we encode the input word. Namely, let $w_1 a w_2$, where $w_1, w_2 \in \Sigma^*$ and $a \in \Sigma$, be the content of the tape at some moment of time, with the working head in the position of the letter a. Denote by w_1^R the reversal of the word w. Then ℓ stores $\mathrm{num}(w_1^R)$, r stores $\mathrm{num}(w_2)$, and a is stored in the finite memory of the underlying finite automaton.

Now, to make a step to the left, we do the following. First, we need to add a to the end of the word encoded by the value of r. This is done by multiplying the value of r by $|\Sigma|+1$ and adding $\mathrm{num}(a)$ to it. Next, we need to extract the last letter of the word encoded by the value of ℓ, and remove this letter. To do so, we do the opposite of what we did for r: this letter is the residue of dividing the value of ℓ by $|\Sigma|+1$, and the new value of ℓ is the result of this division. Notice that all of these operations can be implemented by a fixed VASS as shown in the previous section.

The reason we need the third counter x is to perform these multiplications and divisions. Namely, to divide the value of a counter ℓ by a constant c, we repeat the following until it is no longer possible: subtract c from the value of ℓ and add one to the value of x. When the value of ℓ becomes smaller than c, we get the result of the division in the counter x, and the remainder in ℓ. Multiplication by a constant is done similarly. Observe that by construction the largest value of a counter of $\mathcal{A}$ at any moment of time is at most $(|\Sigma|+1)^S$, where S is the maximal amount of space $\mathcal{M}$ uses on given input. The bit length of this number is linear in S, hence $\mathcal{A}$ simulates $\mathcal{M}$ in linear space. □

By simulating $\mathcal{U}$ from Proposition 35 with a counter automaton $\mathcal{A}$, we get the following statement.

Corollary 37. *There exists a fixed 3-counter automaton $\mathcal{A}$ working in linear space in the bit length of the input such that the zero-reachability problem for it is PSPACE-complete.*

Note that for 2-counter automata, no such result is known. Informally speaking, such automata are exponentially slower than 3-counter automata: the known simulation requires storing the values of the three counters x, y, z as $2^x 3^y 5^z$ [69]. They are also less expressive: for example, 2-counter automata cannot compute the function 2^n [80], while for 3-counter automata this is trivial. It is worth noting the developments of the next subsection imply that a lower bound for fixed 2-counter automata translates into a lower bound for fixed 4-VASS.

4.5.3 From counter automata to VASS

To go from a counter automaton to a VASS, we need to simulate zero tests with a VASS. In general, this is not possible. However, the space complexity of the counter automaton in Corollary 37 is linear, so the values of all its counters are bounded by a polynomial in the bit length of the input. The number of zero tests $\mathcal{A}$ performs does not exceed its time complexity, which is at most exponential in the space complexity. However, this is not a problem, since all the values are provided and stored in binary. The bit length of the number of zero tests is thus polynomial in the input, and hence the testing counters described in Section 4.2.3 can be initialised with a polynomial time reduction, hence obtaining PSPACE-hardness of the zero-reachability problem in fixed 8-VASS.

Moreover, a more advanced technique of quadratic pairs described in [28] allows to deduce the same result for 5-VASS. Namely, a slight variation of [28, Lemma 2.7] states that given a 3-counter automaton $\mathcal{A}$ working in linear space, one can construct a 5-VASS $\mathcal{V}$ such that fixed zero-reachability in $\mathcal{A}$ can be reduced in polynomial time to fixed zero-reachability in $\mathcal{V}$. We briefly explain how we use the technique here. Assume $\mathcal{A}$ has counters $x_1, \ldots, x_d$. We do not use the hatted counters since we know that we construct $\mathcal{A}$ in such a way that the value of the sum $x_1 + \cdots + x_d$ is bounded on any run of $\mathcal{A}$. We introduce counters u_1, u_2 and keep the invariant that $(f(u_1) + f(x_1) + \ldots f(x_d))^2 = f(u_2)$. We use the concept of *flushing* counters. By flushing counter x_1 to x_2 which we represent by $x_1 \to x_2$ we mean the counter program **loop** $\{x_1 -= 1; x_2 += 1; u_2 -= 1\}$. Then, when a zero-test is performed on counter x_1 for example, the following flushes are performed, $u_1 \to x_d \to x_{d-1} \to \ldots \to x_1$ and $u_1 \leftarrow x_d \leftarrow x_{d-1} \leftarrow \ldots \leftarrow x_1$ and we increase the value of u_1 by one and decrease the value of u_2 by one such that the invariant is kept if and only if all the flushes are full and the values of the counter x_1 was zero. After we perform all the necessary zero tests, we only need to check if the final value of u_2 is zero or not.

The same reasoning as before shows that we can initialise the counters of $\mathcal{V}$ to account for enough zero tests. Hence we get the main result of this section.

Theorem 38. *There exists a fixed* 5*-VASS such that the problem of* Fixed VASS zero-reachability *for it is PSPACE-hard, assuming that the input configuration is given in binary.*

4.6 Discussion

In this chapter, we presented an important connection between VASS and arithmetic theories. This is an important step for having a better understanding of the expressivity of VASS. This connection is formalised as Proposition 29 and Proposition 33 which state that given a formula of Short-PA or $\mathbf{FO}k\text{-}\mathbf{aryC}(+,\times)$, there is a counter program that in a sense "models" the formulas. Similarly to the previous chapter, this connection allowed us to discover two new complexity results for the reachability problem when we consider the VASS fixed, the PSPACE lower bound for binary encoding and Theorem 32 for unary encoding. This is a fundamental problem in theoretical computer science, and the setup where the VASS is fixed was a natural open problem that has recently attracted significant attention within the research community. This setup resembles the formalism of database theory, where the database size and the query size are often considered separately. Also, the complexity of the reachability problem with unary inputs turned out to be very low, so we had to design a new type of reduction, in order to be able to analyse the problem and the corresponding language.

We also provided a formalisation of the concept of counter programs. Although the model is quite intuitive, we felt that having some clear semantics defined helped with the more intricate parts of the proofs. We believe that this is a useful formalism that can be used in the search for new lower bounds of other infinite state systems.

Concerning future directions, we think it would be important to also discover the upper bounds of these problems. In the binary case, a recent conjecture of Jecker [51] states that for every VASS $\mathcal{V}$, there exists a fixed constant C such that if a target configuration is reachable from an initial configuration, then there exists a witnessing path whose length is bounded by $C \cdot m$, where m is the maximum constant appearing in the initial and final configurations. Thus, assuming Jecker's conjecture, reachability in fixed VASS under binary encoding of configurations would be PSPACE-complete.

Regarding the lower bounds, one might wonder if we can still retain the PSPACE-hardness for a fixed 3-VASS or 4-VASS. Clearly, we cannot obtain the same hardness

for a fixed 2-VASS since then the reachability set becomes semi-linear. We conjecture that a 5-VASS is needed since the hardness result comes from the simulation power of a 3-counter automaton and we probably need at least 2 extra counters in order to simulate a 3-counter automaton with a VASS. Also, it would be interesting if one could use Theorem 31 in order to obtain a VASS whose reachability problem is complete for any level of the polynomial hierarchy.

In the course of our work, we were not able to find any evidence that this conjecture is false. It is also worth noting that while all our results assume that the final configuration is fixed to a zero vector, we did not find any stronger lower bounds for the case where the final configuration is variable, and only the VASS is fixed. In the unary case, we believe that the concept of arithmetic reductions should be explored further, since multiple problems have the property that the number of inputs is fixed, in which case, it is very hard to discuss the complexity of such problems when the inputs are given in unary. Another question in this area is whether one can relate the complexity of unary FIXED RUDIMENTARY $\mathbf{FO}k\text{-}\mathbf{aryC}(+,\times)$ VALIDITY to other known complexity classes. This might be quite a difficult task, since the unary primality problem is not completely understood yet, but given the general aspect of the problem, one might try to arithmetically reduce FIXED RUDIMENTARY $\mathbf{FO}k\text{-}\mathbf{aryC}(+,\times)$ VALIDITY to other low complexity problems.

Chapter 5

Cost Register Automata

In this chapter, we study the boundedness problem for Cost register automata (CRAs), which are deterministic automata with registers taking values from a fixed semi-ring. As mentioned in the Preliminaries, in Section 2.3.2, CRAs can be seen as VASS with more powerful update functions and registers that can influence each other. All update functions are substitutions over the tropical semiring $\mathbb{K} := (\mathbb{Z} \cup \{+\infty\}, \min, +)$. As such, a CRA computes a function from words to values from $\mathbb{K}$. Given a CRA, the boundedness problem asks if there exists an integer N, such that for every word, the value computed by the CRA on this word is smaller than N. This problem is known to be undecidable for the class of linear CRAs over $\mathbb{K}$, but very little is known about its subclasses. We focus on the subclasses of *copyless* linear CRAs with resets and prove the following results:

1. for copyless linear CRAs with only two registers the problem is NL-complete under unary encoding and coNP-hard under binary encoding,

2. for copyless linear CRAs with an arbitrary number of registers, but when substitutions using minimum can only occur in the outputs, the boundedness problem is PSPACE-complete and if in addition there are no transpositions between registers, then the problem becomes coNP-complete.

We also present a family of CRAs such that the shortest run of value N has a fast-growing (non-elementary with 4+ registers) length. Namely, for each f_i in the hierarchy of fast-growing functions, we provide a stateless CRA with i registers whose output exceeds N only on runs longer than $f_i(N)$. For all of these results, and similarly to Chapter 3, we have to be very careful to analyse all the possible shapes of runs that can lead to a witness of unboundedness. However, this time we also have to deal with the flow of values between different registers. To solve these issues, we look

at NFAs over the alphabet of types of substitutions. We associate with every word over this alphabet a shape that depicts the flow of assignments to registers. Working with this formalism enables us to describe different shapes of witnesses by using the rich language of regular languages.

5.1 Related Work

CRAs are a fairly new computational model, but they are tightly related to well-studied weighted automata (WAs) introduced by Schützenberger in [81], see also surveys [36,37]. In general, CRAs are strictly more expressive than WAs [5]. However, WAs are equally expressive to linear CRAs. Transforming a linear CRA into an equivalent WA and vice versa can be done in polynomial time [5]. Hence, linear CRAs can be seen as a deterministic model for inherently nondeterministic WAs. WAs, and thus CRAs, find their applications in the areas of language and speech processing [70], verification [20], image processing [24], and the analysis of on-line algorithms [7] and probabilistic systems [87]. Functions computable by WAs are exactly those that can be defined in weighted monadic second order logic, a natural extension of monadic second order logic [34,35]. Functions computed by a subclass of CRAs were also characterised in [67] by maximal partition logic, a logic with regular quantifiers that allow to partition words into segments and then aggregate the values computed for them.

To make decision problems tractable for WAs, it is usually required to restrict their expressiveness. The most well-studied way of doing that is by bounding the ambiguity, that is, the number of accepting runs labelled by a word. A WA is called finitely (respectively, linearly, polynomially or exponentially) ambiguous if there exists a constant (respectively, a linear, polynomial or exponential) function $f(n)$ such that for every word w the number of accepting runs labelled by w is bounded by $f(|w|)$. If $f(n) = 1$, a WA is called unambiguous. Most classical decision problems, such as universality, inclusion and equivalence, are undecidable already for linearly ambiguous WAs over the integer min-plus semiring [2, 29, 58]. For finitely ambiguous WAs over this semiring, universality, inclusion and equivalence become decidable [48, 89].

For the boundedness problem, the situation is very different. A seminal paper [2] establishes undecidability of several classical decision problems for linearly ambiguous WAs over the integer min-plus semiring in a uniform fashion. However, it only proves boundedness to be undecidable for general (that is, exponentially ambiguous) WAs,

and provides no classes with decidable boundedness. This indicates that boundedness is somehow different to other mentioned decision problems.

We remark that boundedness is PSPACE-complete for WAs over the *natural* min-plus semiring $(\mathbb{N} \cup \{+\infty\}, \min, +)$ [2, 47, 62, 84], which requires completely different techniques than the integer case. We also remark that it is decidable for copyless linear CRAs over the semiring of positive rational numbers with usual addition and multiplication, and is undecidable for general WAs over the same semiring [26]. Transferring any such results to the min-plus semiring is unlikely, since these semirings has very different properties.

One useful feature of CRAs is that, by adding syntactic restrictions on them, it is possible to introduce subclasses whose expressiveness is incomparable to known classes of WAs. This allows to obtain a finer decidability landscape compared to the case where only the formalism of WAs is used.

One notable example of that is the class of so called copyless linear CRAs. Copyless linear CRAs are strictly less expressive than linearly ambiguous WAs, and their expressivity is incomparable to unambiguous WAs [4]. A further restriction of copyless linear CRAs to the case where the minimum operation is only allowed in the output function makes them equally expressive to finitely sequential WAs, which are unions of WAs whose underlying NFAs are deterministic [6, 30].

Restricting the number of registers in a class of CRAs also usually provides a subclass whose expressivity is incomparable to that of known classes of WAs. For example, there exists a copyless (but not linear) CRA with only 3 registers that computes a function not computed by any polynomially ambiguous WA [68]. For copyless linear CRAs restricted to only three registers universality is undecidable [32]. Moreover, there exist copyless linear CRAs with only two registers that are not equivalent to any unambiguous WA [4], see Figure 2.1 on page 24 for one such CRA. All these results indicate that CRAs with few registers are quite expressive. Results on finding a CRA with the minimum number of registers computing a given function are presented in [30, 31, 54, 55].

5.2 Regular substitution languages

In the boundedness problem, the input alphabet of a CRA is redundant, since the formulation is existentially quantified for the word and the underlying finite automaton is deterministic. For this reason, in this paper we focus on sequences of substitutions that a CRA can perform.

A *language of substitutions* is an arbitrary subset of $\mathrm{Sub}_{\mathrm{clr}}(X)^*$. Every word $w \in \Sigma^*$ read by a CRA $\mathcal{C}$ induces a sequence of substitutions that $\mathcal{C}$ performs when reading w. Fix a CRA $\mathcal{C} = \langle X, \Sigma, Q, q_{\mathrm{ini}}, \delta, \mathrm{out}\rangle$. We define the language of substitutions $\mathcal{L}_x(\mathcal{C})$ induced by it. Observe that the set of substitutions that occur in the transitions and output expressions of $\mathcal{C}$ is finite. We denote it by $\Gamma_{\mathcal{C},x}$. The regular language $\mathcal{L}_x(\mathcal{C})$ is then formally defined as the language of the following automaton $\mathrm{NFA}_x(\mathcal{C})$. To simplify the presentation, we assume that the last substitution in the sequence corresponding to a word saves the output into a designated output register $x \in X$.

Definition 39 ($\mathrm{NFA}_x(\mathcal{C})$)**.** *Given a CRA $\mathcal{C} = \langle X, Q, \Sigma, \delta, q_{\mathrm{ini}}, \mathrm{out}\rangle$, define a nondeterministic finite automaton $\mathrm{NFA}_x(\mathcal{C}) \coloneqq \langle Q', \Gamma_{\mathcal{C},x}, \delta', q_{\mathrm{ini}}, \{q_{\mathrm{fin}}\}\rangle$ where $Q' \coloneqq Q \cup \{q_{\mathrm{fin}}\}$ and the transition relation $\delta' \subseteq Q' \times \Gamma_{x,\mathcal{A}} \times Q'$ contains transitions:*

$$(q, \nu, q') \in \delta' \qquad \text{for every } q, q' \in Q, \text{ and } \nu, \sigma \text{ such that } \delta(q, \sigma) = (q', \nu),$$
$$(q, \{x \leftarrow \mathrm{out}(q)\}, q_{\mathrm{fin}}) \in \delta' \qquad \text{for every } q \in Q.$$

Fix an NFA $\mathcal{A} = (Q, S, \delta, q_{\mathrm{ini}}, Q_{\mathrm{fin}})$ and a set of registers $X = \{x_1, \ldots, x_d\}$ for the rest of this subsection. An alternating sequence $\pi = q_0 \xrightarrow{\nu_1} q_1 \xrightarrow{\nu_2} q_2 \to \cdots \to q_{n-1} \xrightarrow{\nu_n} q_n$ of states from Q and letters from S is called *a run in $\mathcal{A}$ labelled by the word $w = \nu_1\nu_2\cdots\nu_n$*. We write $q_0 \xrightarrow{\pi,w}\!\!\!\!\!\rightarrow\, q_n$ to denote the fact that π is a run labelled by w that begins in q_0 and ends in q_n. For such a run and $\mu \in \mathrm{Val}(X)$, we define $\mathrm{eval}_\mu(\pi) \coloneqq \mathrm{eval}_\mu(w)$. We identify words with compositions of the corresponding sequences of substitutions. The set of runs of $\mathcal{A}$ is denoted by $\mathrm{Runs}(\mathcal{A})$. A run π is *accepting* if $q_{\mathrm{ini}} \xrightarrow{\pi}\!\!\!\!\!\rightarrow\, q_{\mathrm{fin}}$. An NFA $\mathcal{A}$ *accepts* $w \in S^*$ whenever there exists an accepting run of $\mathcal{A}$ labelled by w. The *language of* $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$.

To be able to refer to segments of runs, we combine the notation for single transitions $p \xrightarrow{\nu} q$ and runs $q \xrightarrow{\pi,w}\!\!\!\!\!\rightarrow\, r$. For example, we may consider a run $\pi = p \xrightarrow{\nu} q \xrightarrow{\pi',w}\!\!\!\!\!\rightarrow\, r$ labelled by a word $\nu \cdot w$. When the labelling is not important, we write $q \xrightarrow{\pi}\!\!\!\!\!\rightarrow\, r$.

We mix the notation $q \xrightarrow{\nu} q'$ and $q \xrightarrow{\pi,w}\!\!\!\!\!\rightarrow\, q'$ to introduce named segments of runs, when it does not introduce any ambiguity.

Example 40 (Named segments of a run)**.** *Consider the following notation:*

$$\pi = p \xrightarrow{\nu} q \xrightarrow{\pi',w}\!\!\!\!\!\rightarrow\, r$$

It means that π is composed of a single transition over ν followed by a sub-run π' from q to r; $\pi = p\nu\pi'$; it is over the word $\nu \cdot w$.

There is an obvious correspondence between runs in $\mathrm{eval}(\mathcal{C})$ and in $\mathrm{NFA}_x(\mathcal{C})$. In particular, for $k \in \mathbb{K}$, there exists $w \in \Sigma^*$ such that $\mathcal{C}(w) = k$ if, and only if, there exists $u \in \mathcal{L}(\mathcal{A})$ such that $\mathrm{eval}_{\mathbf{0}}(u)(x) = k$. This allows us to restate the boundedness problems in terms of languages of substitutions. We say that a regular language $L \subseteq \mathrm{Sub}_{\mathrm{clr}}(X)^*$ *has bounded output* in $x \in X$ if there exists $N \in \mathbb{N}$ such that for every $w \in L$ we have $\mathrm{eval}_{\mathbf{0}}(x) < N$.

Problem 41. BOUNDEDNESS OF REGULAR d-REGISTER SUBSTITUTION LANGUAGES
Input: *NFA $\mathcal{A}$ over a finite set $S \subset \mathrm{Sub}_{\mathrm{clr}}(X)$, $|X| = d$, output register $x \in X$.*
Output: *Yes if and only if $\mathcal{L}(\mathcal{A})$ has bounded output in x.*

Note that boundedness for CRAs (Problem 9) easily reduces to this problem in deterministic logarithmic space. Indeed, it suffices to compute NFA $\mathcal{A}$ for a given CRA with Definition 39. This reformulation allows us to use a rich framework of regular languages, which streamlines the proofs presented in later sections.

Next, we aim to decompose the linear copyless substitutions. When discussing sections of a run of CRA it is useful to distinguish between a substitution that has the effect of only adding a constant to a register, and one that transfers values from one register to another, via a minimum operation.

Definition 42 (Elementary substitutions)**.** *We say that a substitution $\nu \in \mathrm{Sub}_{\mathrm{clr}}(X)$ is* elementary *if it has one of the following forms for some $x, y \in X$, $c \in \mathbb{K}$:*

$$\{x \leftarrow x + c\} \quad \text{(additive sub.)} \qquad \{x \leftarrow y,\ y \leftarrow x\} \quad \text{(transposition)}$$
$$\{x \leftarrow \min\{x, y\},\ y \leftarrow 0\} \quad \text{(minimum sub.)} \qquad \{x \leftarrow 0\} \quad \text{(reset sub.)}$$

Let $\mathrm{Sub}_{\mathrm{elem}}(X) \subseteq \mathrm{Sub}_{\mathrm{clr}}(X)$ be the set of elementary substitutions, and let $T_X \cup R_X \cup A_X \cup M_X$ be its partition of $\mathrm{Sub}_{\mathrm{elem}}(X)$ into sets of transpositions, and reset, additive, and minimum substitutions, respectively.

The next lemma allows us to express copyless linear substitutions with resets in the form of multiple elementary substitutions.

Lemma 43. *For every $\nu \in \mathrm{Sub}_{\mathrm{clr}}(X)$ in a canonical form, there exists a word of substitutions $u \in \mathrm{Sub}_{\mathrm{elem}}(X)^*$ of length $O(d^2)$ such that $u \equiv \nu$.*

Proof. Take arbitrary X of size $k \in \mathbb{N}$. Fix $\nu \in \mathrm{Sub}_{\mathrm{clr}}(X)$. Let $\sigma \colon X \rightharpoonup X$ be a partial function satisfying $\sigma(x) = y$ if $\nu(x)$ has the form $\min\{y + c_1, \dots\}$, i.e. , $\nu(x)$ is linear and the first register from the left appearing in it is y. Note that σ is injective due to

copylessness, therefore it can be extended to a permutation $\sigma' \colon X \to X$. Naturally, σ' is expressible as a composition of $m \leq k^2$ transpositions. Therefore, there exists a sequence of m transposition substitutions $u \in T_X^m$ such that $\nu' \coloneqq \nu; \tau$ always assigns expressions of the form $\min\{y + c_1, \dots\}$ to y. Note that $\nu = \nu'; u^{\mathrm{rev}}$, where u^{rev} is the reverse of u.

Finally, every mapping $(x_1 \leftarrow \min\{x_1 + c_1, x_2 + c_2, \dots, x_n + c_n\}) \in \nu'$ $(n \leq k)$ can be realised by a sequence of length $2n = O(k)$:

$$\{x_1 \leftarrow x_1 + c_1\}; \{x_2 \leftarrow x_2 + c_2\}; \dots; \{x_n \leftarrow x_n + c_n\};$$
$$\{x_1 \leftarrow \min\{x_1, x_2\}\}; \{x_1 \leftarrow \min\{x_1, x_3\}\}; \dots; \{x_1 \leftarrow \min\{x_1, x_n\}\}$$

and there are at most k of them, yielding a sequence of simple substitutions of length $O(k)$ equivalent to ν, which completes the proof. □

Note that the statement of the above lemma is not true in the general setting of $\mathrm{Sub}(X)$, as its proof relies on copylessness. Note also that Lemma 43 implies the existence of a homomorphism to-elem$\colon \mathrm{Sub}_{\mathrm{clr}}(X)^* \to \mathrm{Sub}_{\mathrm{elem}}(X)^*$ such that $\nu \equiv$ to-elem(ν) for every $\nu \in \mathrm{Sub}_{\mathrm{clr}}(X)$. For a finite set $S \subset \mathrm{Sub}(X)$, we define $\mathrm{maxc}(S) \coloneqq \max\{\mathrm{maxc}(\nu)$ such that $\nu \in S\}$. The following claim is immediate.

Claim 44 (Maximal constant grows linearly with respect to length)**.** *Fix a finite set of substitutions $S \subset \mathrm{Sub}_{\mathrm{elem}}(X)$. For every $w \in S^*$, we have* $\mathrm{maxc}(w) \leq |w| \cdot \mathrm{maxc}(S)$.

Finally, we rely on the following claim to restrict the alphabet to only elementary subsitutions.

Claim 45 (Elementary substitutions assumption)**.** *We may assume without loss of generality that the alphabet S of $\mathcal{A}$ consists only of elementary substitutions,* i.e.*, $S \subset \mathrm{Sub}_{\mathrm{elem}}(X)$.*

Proof. Building on Lemma 43, we show that for any NFA $\mathcal{A}$ over $S \subset \mathrm{Sub}_{\mathrm{clr}}(X)$ there exists an NL-constructible NFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = $ to-elem$(\mathcal{L}(\mathcal{A}))$. Indeed, the construction only requires subdividing every transition $p \xrightarrow{\nu} s$ in $\mathcal{A}$ into

$$p \xrightarrow{\nu'_1} q_1 \xrightarrow{\nu'_2} q_2 \to \cdots \to q_{n-1} \xrightarrow{\nu'_n} r\,,$$

where $\nu'_1 \nu'_2 \cdots \nu'_n =$ to-elem ν, 'q_i's are fresh states, and $n = O(1)$. □

5.2.1 The structure of witnesses with additive and reset substitutions only

In this subsection, we show how to simplify and decompose runs that do not contain any minimum substitutions or transpositions. We then use these results in the complexity upper bounds and to analyse runs with a more complex structure.

For the rest of this subsection, fix a set of registers $X = \{x_1, \dots, x_d\}$, an output register $x \in X$, and an NFA $\mathcal{A}$ over a finite alphabet $S \subset A_X \cup R_X \subset \mathrm{Sub}_{\mathrm{elem}}(X)$ of elementary additive and reset substitutions. Similarly to Claim 45, this covers a more general case where the alphabet of $\mathcal{A}$ consists of substitutions adding integer values to some registers and resetting other registers.

Let $w \in A_X^*$ be such that $w \equiv \{x_1 \leftarrow x_1 + c_1, \dots, x_d \leftarrow x_d + c_d\}$ for some $c_1, \dots, c_d \in \mathbb{K}$. We define $\mathrm{eff}(w) \coloneqq (c_1, \dots, c_d) \in \mathbb{K}^X$, and we call it the *effect* of w. The *integer conic hull* $\mathrm{Cone}_{\mathbb{N}}(V)$ of a set of vectors V is the set of linear combinations of vectors from V with nonnegative integer coefficients. The following theorem is a direct consequence of a classical result by Carathédory, see, *e.g.*, [79].

Theorem 46 (Only d vectors are sufficient to represent a positive point)**.** *Let $V \subseteq \mathbb{Z}^d$. If all components of a vector $\boldsymbol{b}$ are strictly positive and $\boldsymbol{b} \in \mathrm{Cone}_{\mathbb{N}}(V)$, then there exists $V' \subseteq V$ with $|V'| \leq d$ and a constant $\lambda > 0$ such that $\lambda \boldsymbol{b} \in \mathrm{Cone}_{\mathbb{N}}(V')$.*

Claim 47. *Let $V \in \mathbb{Q}^X$. If $\boldsymbol{b} > \mathbf{0}$ and $\boldsymbol{b} \in \mathrm{Cone}(V)$ then there exists $V' \subseteq V$ and $\boldsymbol{b}' > \mathbf{0}$ such that $\boldsymbol{b}' \in \mathrm{Cone}_{\mathbb{N}}(V')$ and $|V'| \leq d$.*

We aim to use this result in order to obtain a small representation of long runs. Intuitively, the effect of an additive cycle can be described by an integer vector, and thus, the effect of a run that has many additive cycles can be described as a vector inside the integer cone of the "cycle vectors". Thus, we can argue that we only need a small number of these cycles in order to obtain a more compact run with the same effect. This intuition is formalised by the following lemmas describing the shape of runs labelled by words over A_X and $A_X \cup R_X$, respectively.

Lemma 48 (Decomposition lemma for additive runs)**.** *For every run $q \xrightarrow{\pi, w} q'$ of $\mathcal{A}$ such that $w \in A_X^*$, there exist $n \in \mathbb{N}$, words $w_1, \dots, w_n, , w', z_1, \dots, z_{n+1} \in A_X^*$, and integers $a_1, \dots, a_n \in \mathbb{N}$ such that*

- *for every word $z \in z_1 w_1^* z_2 \dots z_n w_n^* z_{n+1}$, there exists a run $q \xrightarrow{z} q'$,*
- $\mathrm{eff}(w) = \mathrm{eff}(w') + a_1 \mathrm{eff}(w_1) + \dots + a_n \mathrm{eff}(w_n)$, *and*

- $|w_1|, \ldots, |w_n|, |w'| \leq |Q|$ *and* $|z_1 \ldots z_{n+1}| \leq |Q|^2$.

The idea of the proof is that we iterate a process that eliminates all the simple cycles from π. These simple cycles are labelled by some words $w_1, \ldots, w_d$ and the process returns a cycle-free run π' that is labelled by a word w'. We can describe the additive effect of the run π as the effect of w' plus the effect of all the simple cycles that we eliminated. Finally, we argue there exists a short run from q to q' that contains a vertex from each of these simple cycles.

Proof. Consider the following iterative process on π. Initialise $\boldsymbol{b} = \mathbf{0} \in \mathbb{Q}^X$.

- Identify the first simple cycle $r \xrightarrow{\pi_i, w_i} r$ in π and replace it by r. By simple cycle we mean a run where only the first and last states are equal.
- Update the value of the vector $\boldsymbol{b}$ to $\boldsymbol{b} + \operatorname{eff}(w_i)$.

Let $q \xrightarrow{\pi', w'} q'$ be the resulting run. This process guarantees that π' is cycle-free and that $\operatorname{eff}(w) = \operatorname{eff}(w') + \boldsymbol{b} = \operatorname{eff}(w') + a_1 \operatorname{eff}(w_1) + \cdots + a_n \operatorname{eff}(w_n)$, where a_i is the number of times we have eliminated a simple cycle with effect w_i, for all $1 \leq i \leq n$.

Let $Q' \subseteq Q$ be the set of states visited by π. The shortest run that visits all states of Q' has length at most $|Q|^2$. Since every word w_i corresponds to a simple cycle eliminated by the process, it follows that there exists words $z_1, \ldots, z_{n+1}$ such that for every word $z \in z_1 w_1^* z_2 \ldots z_n w_n^* z_{n+1}$, there exists a run $q \xrightarrow{z} q'$ and $|z_1 \ldots z_{n+1}| \leq |Q|^2$ even if n can be as large as $2^{|Q|}$. □

Lemma 49 (Pumping lemma)**.** *If $S \subseteq A_X \cup R_X$, then there exists a word $w \in \mathcal{L}(\mathcal{A})$ with $\operatorname{eval}_{\mathbf{0}}(w) > 2d\,|Q|\,C$, where $C \coloneqq \operatorname{maxc}(S)$ if and only if there exist words $\alpha_1, \ldots, \alpha_{d+1}$, $\beta_1, \ldots, \beta_d \in A_X^*$ such that*

- $\alpha_1 \beta_1^+ \alpha_2 \ldots \alpha_d \beta_d^+ \alpha_{d+1} \eta_X \subseteq \mathcal{L}(\mathcal{A})$,
- $|\beta_1|, \ldots, |\beta_d| \leq |Q|$
- $|\alpha_1 \ldots \alpha_{d+1}| < (d+1)\,|Q|^2$, *and*
- *for each $N \in \mathbb{N}$, there are $a_1, \ldots, a_d \in \mathbb{N}$ with $\operatorname{eval}_{\mathbf{0}}(\alpha_1 \beta_1^{a_1} \alpha_2 \ldots \alpha_d \beta_d^{a_d} \alpha_{d+1} \eta_X)(x_1) > N$.*

Proof idea. Given a run in $\mathcal{A}$ of sufficiently large value, we can split it into different segments such that in each segment we know for every register if it is going to be reset in the future or not. Thus, for every register, we can determine if a segment of

the run is relevant for determining its final value. Subsequently, we identify all cycles of this run and argue that since all the registers hold large values at the end of the run, the total effect of the relevant cycles (the ones after the last reset of the register) must be a large positive number for each register. This allows us to use Lemma 48 to conclude that we do not have too many such cycles and that we can pump them up in order to achieve unbounded register values. □

Proof. The right to left implication of this lemma is immediate, so we focus on the left to right implication. Assume there exists a run $\pi = q_{\text{ini}} \xrightarrow{w} q_{\text{fin}}$, for some $q_{\text{fin}} \in Q_{\text{fin}}$ such that $\text{eval}_{\mathbf{0}}(w)(x_1) > 2d\,|Q|\,C$.

We can assume that the value of each register is reset at least once along π, since we can add additional reset substitutions to the beginning of w without changing the value of $\text{eval}_{\mathbf{0}}(w)(x_1)$. For each i, $1 \leq i \leq d$, the ν_i be the last reset substitution for register x_i in w. Without loss of generality, we can assume that in w, the registers are reset for the last time in the increasing order of their indices. Then, π can be represented as

$$\pi = q_{\text{ini}} \xrightarrow{\pi_1, w_1} q_1 \xrightarrow{\nu_1} q_1' \xrightarrow{\pi_2, w_2} q_2 \xrightarrow{\nu_2} \dots \xrightarrow{\pi_d, w_d} q_d \xrightarrow{\nu_d} q_d' \xrightarrow{\pi_{d+1}, w_{d+1}} q_{d+1} \xrightarrow{\eta_X} q_{\text{fin}}\,.$$

Observe that for each i, $1 \leq i \leq d-1$, we can replace in w_i all reset substitutions of registers $x_{i+1}, \dots, x_d$ with the identity substitution without changing the value of $\text{eval}_{\mathbf{0}}(w)(x_1)$. Thus, $w_1, \dots, w_{d+1}$ become words over additive substitutions only. Hence we now get that $\text{eval}_{\mathbf{0}}(w_1\nu_1 \cdots w_{d+1}) = \text{eff}(w_1) + \cdots + \text{eff}(w_{d+1})$. By applying Lemma 48 to each run $\pi_1, \dots, \pi_{d+1}$, we obtain that

$$\begin{aligned} \text{eval}_{\mathbf{0}}(w_1\nu_1 \dots w_{d+1}) = \; & \text{eff}(w_1') \; + \; a_1^{(1)}\text{eff}\left(w_1^{(1)}\right) \; + \cdots + \; a_{n_1}^{(1)}\text{eff}\left(w_{n_1}^{(1)}\right) \\ & \vdots \qquad\qquad \vdots \qquad\qquad\qquad \vdots \\ & \text{eff}(w_{d+1}') + a_1^{(d+1)}\text{eff}\left(w_1^{(d+1)}\right) + \cdots + a_{n_{d+1}}^{(d+1)}\text{eff}\left(w_{n_{d+1}}^{(d+1)}\right) \end{aligned}$$

where $w_1^{(1)}, \dots, w_{n_{d+1}}^{(d+1)}, w_1', \dots, w_{d+1}'$ are all words of length smaller than $|Q|$. Since

$$\text{eval}_{\mathbf{0}}(w_1\nu_1 \cdots w_{d+1})(x) > 2d\,|Q|\,C \quad \text{and} \quad \text{eff}\left(w_1' \cdots w_{d+1}'\right)(x) \leq (d+1)\,|Q|\,C$$

for every $x \in X$, there must exists a vector $\boldsymbol{b} \in \mathbb{N}_+^X$ such that $\boldsymbol{b} \in \text{Cone}_{\mathbb{N}}\left(\left\{w_1^{(1)}, \dots, w_{d+1}^{(d+1)}\right\}\right)$. By Theorem 46, there exists a subset $\{\beta_1, \dots, \beta_d\} \subseteq \left\{w_1^{(1)}, \dots, w_{d+1}^{(d+1)}\right\}$ such that $\boldsymbol{b}' \in \text{Cone}_{\mathbb{N}}(\{\beta_1, \dots, \beta_d\})$ and $\boldsymbol{b}' > \mathbf{0}$ (pointwise).

It remains to show that there exist words $\alpha_1, \dots, \alpha_{d+1}$ such that $|\alpha_1 \dots \alpha_{d+1}| < (d+1)\,|Q|^2$ and $\alpha_1\beta_1^+\alpha_2 \dots \alpha_d\beta_d^+\alpha_{d+1}\eta_X \in \mathcal{L}(\mathcal{A})$. Indeed, consider a run π_i for some

$1 \leq i \leq d+1$. Either π_i contains no cycles from $\{\beta_1, \ldots, \beta_d\}$, in which case there exists a word w'_i of length at most $|Q|$ corresponding to the run π'_i, or it does contain some cycles $\{\beta_j, \ldots, \beta_{j+h}\}$ from $\{\beta_1, \ldots, \beta_d\}$. In the latter case, we can use Lemma 48 to argue that there exist words $z'_j, \ldots, z'_{j+h+1}$ such that for every word $z \in z'_j \beta_j^* z'_{j+1} \ldots z'_{j+h} \beta_{j+h}^* z'_{j+h+1}$, there exists a run $q'_{i-1} \xrightarrow{z} q_i$. In both cases, either $|w'| \leq |Q|$ or $\left|z'_j \ldots z'_{j+h+1}\right| \leq |Q|^2$. Thus, we can obtain the required words $\alpha_1, \ldots, \alpha_{d+1}$ by possibly concatenating the w'_i and z'_i words that we identified. □

5.3 CRAs with two registers

In this section, we prove the following result.

Theorem 50. *The boundedness problem for CRAs with two registers is NL-complete if the numbers in the substitutions are presented in unary, and in coNP if they are in binary.*

By the results of the previous section, this theorem is equivalent to the following statement.

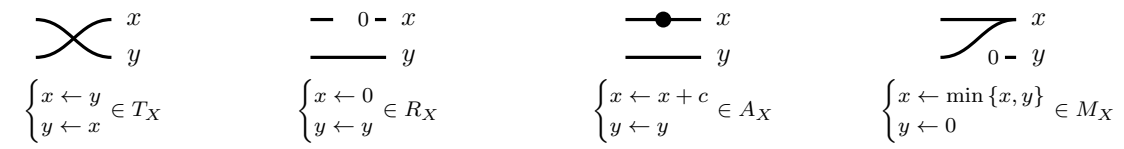
Proposition 51. *Boundedness of regular 2-register substitution languages is NL-complete if the numbers in the substitutions are presented in unary, and in coNP if they are in binary.*

The remainder of the section is devoted to proving Proposition 51. For the rest of this section, fix the set of registers $X \coloneqq \{x, y\}$ and the output register x. Let us also fix for the rest of the section an input NFA $\mathcal{A} = \langle Q, S, \delta, q_{\text{ini}}, Q_{\text{fin}} \rangle$ such that $S \subset T_X \cup A_X \cup M_X \cup R_X \subset \text{Sub}_{\text{elem}}(X)$ (which we can assume by Claim 45). We begin by providing a series of simplifying assumptions for the automaton $\mathcal{A}$ in the input.

In this subsection, we introduce a normal form that significantly simplifies our arguments.

Definition 52 (Graphical notation for $\text{Sub}_{\text{elem}}(X)$)**.** *We depict elementary substitutions in* $\text{Sub}_{\text{elem}}(X)$ *as shown in Figure 5.1; details of the notation are discussed in its caption.*

For a word $w = \nu_1 \nu_2 \cdots \nu_n \in \text{Sub}_{\text{elem}}(X)^*$, we define the *output derivation tree* out-tree(w) as the derivation tree of the expression $w(x)$. This tree can have three kinds of leaves: constants 0 originating from reset or minimum substitutions, arbitrary



(a) transposition (b) reset substitution (c) additive substitution (d) minimum substitution

Figure 5.1: Pictorial representation of elementary substitutions in $\mathrm{Sub}_{\mathrm{elem}}(\{x, y\})$. Register x is always drawn above y. A black dot stands for adding a constant – our graphical notation disregards the particular values of constants in the substitutions. A branching depicts the minimum operation. For the operations on y, the diagrams are symmetric. Diagrams are mirrored vertically when we add to y, reset it, or store the result of a minimum in it. The effect of gluing several drawings together horizontally naturally corresponds to composition of depicted substitutions.

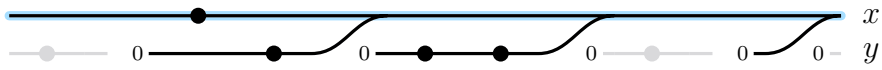
constants from $\mathbb{K}$ coming from additive substitutions of the form $\{r \leftarrow r + c\}$ (for $c \in \mathbb{K}, r \in X$), and variables occurring in $\nu_1(x)$ or $\nu_1(y)$ of the first substitution ν_1. We say that a leaf has level i if it comes from substitution ν_i. We say that a path from a leaf to the root in out-tree(w) is *leading* if its starting leaf comes from a substitution ν_i with the smallest i among all leaves that have a path to the root which has the minimal level. A path is called *x-aligned* if all its vertices originate from expressions $(\nu_i(x))_{1 \le i \le n}$. A word w is called *x-aligned* if the leading path of out-tree(w) is x-aligned.

Example 53 (Depiction of out-tree(w))**.** *For $w \in \mathrm{Sub}_{\mathrm{elem}}(X)^*$,* out-tree$(w)$ *corresponds to the tree rooted at the rightmost position corresponding to x in the pictorial representation. Consider*

$$w := \begin{cases} x \leftarrow x \\ y \leftarrow y + 3 \end{cases} \begin{cases} x \leftarrow x \\ y \leftarrow 0 \end{cases} \begin{cases} x \leftarrow x + 2 \\ y \leftarrow y \end{cases} \begin{cases} x \leftarrow x \\ y \leftarrow y + 2 \end{cases} \begin{cases} x \leftarrow \min\{x, y\} \\ y \leftarrow 0 \end{cases} \begin{cases} x \leftarrow y \\ y \leftarrow x \end{cases} \begin{cases} x \leftarrow x + 3 \\ y \leftarrow y \end{cases}$$
$$\begin{cases} x \leftarrow x + 3 \\ y \leftarrow y \end{cases} \begin{cases} x \leftarrow \min\{x, y\} \\ y \leftarrow 0 \end{cases} \begin{cases} x \leftarrow x \\ y \leftarrow y + 5 \end{cases} \begin{cases} x \leftarrow x \\ y \leftarrow 0 \end{cases} \begin{cases} x \leftarrow \min\{x, y\} \\ y \leftarrow 0 \end{cases}$$

The depiction of w, in line with Definition 52, is as follows:

The out-tree(w) *corresponds to the subgraph drawn in black. The leading path π of* out-tree(w) *is marked with a blue outline. Expressions in w corresponding to vertices of π are typeset on blue background. As not all of them come from $x \leftarrow e$ mappings, path π is not x-aligned. An x-aligned word w' such that $w(x) \equiv w'(x)$ has the following shape:*



Lemma 54 (Leading branch x-aligned assumption)**.** *We can assume that each $w \in \mathcal{L}(\mathcal{A})$ is x-aligned.*

Proof. Fix a finite alphabet $S \subset \mathrm{Sub}_{\mathrm{elem}}(X)$. We close S with respect to swapping x and y: let $S_{\diagdown\!\!\diagup} = S \cup \{x \leftarrow \nu(y),\ y \leftarrow \nu(x) \text{ such that } \nu \in S\}$. First, observe that

Claim 55. *For any 2-register elementary substitution word $w \in S^*$ there exists an x-aligned $w' \in S_{\diagdown\!\!\diagup}^*$ such that $w(x) \equiv w'(x)$ and w' does not feature any transpositions.*

The simple yet technical proof of this claim is omitted, cf. Example 53. Intuitively, it is enough to swap the registers and propagate this swap to straighten out the leading branch.

Claim 56. *The language of all x-aligned words equivalent to words in $\mathcal{L}(\mathcal{A})$ is regular, and the corresponding NFA can be computed in deterministic logarithmic space.*

It is easy to define the language L of x-aligned words over $S_{\diagdown\!\!\diagup}$. Indeed, since regular languages are closed with respect to reverse operation, we can start traversing the word from the end and keep track of reachable nodes. It suffices to verify that the path from the root to the final reachable node was x-aligned. Similarly easily, we can define the language $\mathcal{L}_{\diagdown\!\!\diagup}(\mathcal{A})$ of words w' which arise from $w \in \mathcal{L}(\mathcal{A})$ by vertically mirroring parts of their depictions. Intersection $L \cap \mathcal{L}_{\diagdown\!\!\diagup}(\mathcal{A})$ contains all x-aligned words equivalent to words from $\mathcal{L}(\mathcal{A})$ and an automaton recognising it is NL-constructible from $\mathcal{A}$. □

Therefore, in the remainder of this section we assume that each $w \in \mathcal{L}(\mathcal{A})$ is x-aligned. This means that only allow additive substitutions, reset substitutions only for y, and minimum substitutions only for x.

5.3.1 Unboundedness witnesses

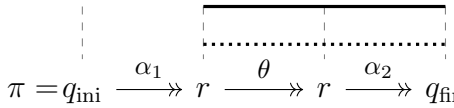
In this subsection, we define the shape of witnesses that prove unboundedness of CRAs. Next subsection shows that if a CRA is unbounded, it must contain such a witness. For the rest of this section, we fix two substitutions

$$\eta = \{x \leftarrow \min\{x, y\},\ y \leftarrow 0\} \text{ and } \rho = \{x \leftarrow x,\ y \leftarrow 0\},$$

which will be referred to throughout the whole section. Let $N \coloneqq |Q|$ and $C \coloneqq \mathrm{maxc}(S)$. We introduce two new notations for special types of runs of $\mathcal{A}$:

used to signify that $w_1 \in A_X^*$ (*i.e.*, has additive substitutions only), and $w_2 \in (A_X \cup \{\rho\})^*$.

Definition 57 (Unboundedness witness)**.** *A run π in $\mathcal{A}$ is called a* trivial unboundedness witness *if it has the form*



such that $|\pi| \leq 3N$, $\text{eff}(\theta)(x) > 0$ and $q_{\text{fin}} \in Q_{\text{fin}}$.

A run π in $\mathcal{A}$ is called a nontrivial unboundedness witness *if it has the form*

$$\pi = q_{\text{ini}} \xrightarrow{\pi_a} q_a \xrightarrow{\pi_b} q_b \xrightarrow{\pi_c} q_c$$

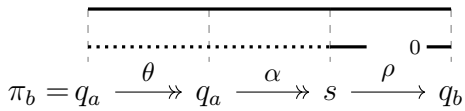
such that $|\pi_a| \leq N$, $|\pi_b|, |\pi_c| \leq 3N^2$, run π_b is pumpable, and π_c is sustainable, where pumpable and sustainable runs are defined below.

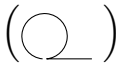
A trivial or nontrivial unboundedness witness is called just an unboundedness witness*.*

Intuitively, π_a from this definition is used to reach a gadget enabling pumping, π_b witnesses that we can pump up the value of x to an arbitrarily large number and then end up in q_b, and π_c certifies that we can maintain a large value of x to be output in $q_c \in Q_{\text{fin}}$.

Definition 58 (Pumpable run)**.** *A run π_b is called* pumpable *if it has one of the four forms:*

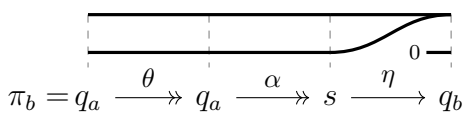
- **Type A.1** (a cycle with a positive effect on x, then a reset of y)

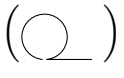
  

  such that θ is a cycle with $\text{eff}(\theta)(x) > 0$ and α is a run with no η substitutions.

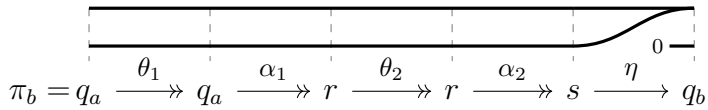
- **Type A.2** (a cycle with a positive effect on both x and y, then a minimum substitution)

  such that θ is a cycle with no η and no ρ substitutions and with $\text{eff}(\theta) \in \mathbb{N}_+^2$, and α is a run with no η and no ρ substitutions.

- **Type A.3** (two cycles combining for a positive effect on both x and y, then a minimum)

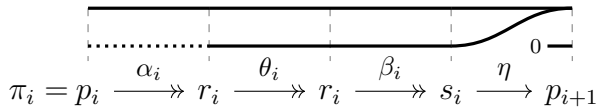


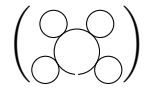
such that θ_1, θ_2 are cycles with no η and no ρ substitutions and with $a_1\mathrm{eff}(\theta_1) + a_2\mathrm{eff}(\theta_2) \in \mathbb{N}^2_+$, for some $a_1, a_2 \in \mathbb{N}$. Furthermore, α_1, α_2 are runs with no η and no ρ substitutions.

- **Type B** (a cycle with a positive effect on x together with cycles supporting its value)

$$\pi = p_1 \xrightarrow{\pi_1} p_2 \xrightarrow{\pi_2} p_3 \twoheadrightarrow \cdots \twoheadrightarrow p_{n-1} \xrightarrow{\pi_{n-1}} p_n \xrightarrow{\pi_n} p_1$$

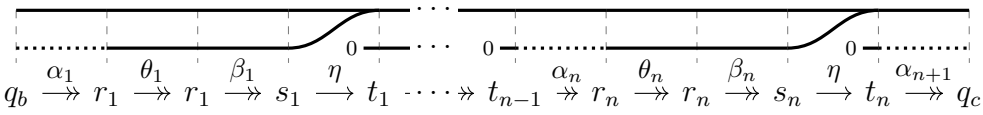
for some $n \in \mathbb{N}$, where $p_1 = q_a = q_b$, and for every i, $1 \leq i \leq n$:





such that α_i is a run with no η substitutions, θ_i is a cycle with no η and no ρ substitutions with $\mathrm{eff}(\theta_i) \in \mathbb{N} \times \mathbb{N}_+$ and β_i is a run with no η substitutions. Furthermore, we require $\mathrm{eff}(\alpha_1\theta_1\beta_1\alpha_2\theta_2\beta_2 \cdots \alpha_n\theta_n\beta_n)(x) > 0$.

Definition 59 (Sustainable run)**.** *A run π_c is called* sustainable *if it is labelled by $w_c \in (A_X \cup \{\rho, \eta\})^*$ and has the following form:*



for $n \in \mathbb{N}$, runs α_i, β_i of the form as depicted in the picture, and cycles θ_i such that $\mathrm{eff}(\theta_i) \in \mathbb{Z} \times \mathbb{N}_+$ for every i, $1 \leq i \leq n$.

Proposition 60. *Given $\mathcal{A}$, deciding if there exists a run in it which is an unboundedness witness is in NL if the numbers in the substitutions are presented in unary, and in NP if they are in binary.*

Proof. Intuitively, using nondeterminism, we can guess a nontrivial witness $\pi = \pi_a\pi_b\pi_c$ as in Definition 57 and verify that it has all the required properties. The case of a trivial witness is handled in a similar way and is thus omitted.

Indeed, starting in q_{ini}, we guess one transition at a time until we verify the existence of a witness or exceed the bound $N + 4N^2$ on its length. During the traversal, we guess the positions of states q_a, q_b, q_c at appropriate distances from q_{ini}. This splits the search into three phases corresponding to π_a, π_b and π_c. We need to verify that π_b is of one of four types of pumpable runs (cf. Definition 58). At any point in time, if the definition of a pumpable run requires it, we can guess that the current state r marks the start of the occurrence of a simple cycle θ. In this case, we store r, follow only labels from $A_X \cup \{\rho\}$ and compute the effect of the run until r occurs again. This can easily be done in NL or NP depending on the representation of the numbers in the substitutions. □

In order to complete the proof of Proposition 51, we need to show that the existence of an unboundedness witness is equivalent to the fact that $\mathcal{A}$ is not bounded. We show it by two implications stated below. We start with proving Lemma 61, and Lemma 64 is proved in the next subsection.

Lemma 61. *If there exists an unboundedness witness then $\mathcal{L}(\mathcal{A})$ is not bounded.*

Proof. The proof is straightforward in case of a trivial unboundedness witness. Fix a nontrivial unboundedness witness π as in Definition 57:

$$\pi = q_{\text{ini}} \xrightarrow{\pi_a}\!\!\!\!\!\rightarrow\; q_a \xrightarrow{\pi_b}\!\!\!\!\!\rightarrow\; q_b \xrightarrow{\pi_c}\!\!\!\!\!\rightarrow\; q_c\,.$$

Fix an arbitrary number $N \in \mathbb{N}$. We construct a run π' such that $\text{eval}_{\mathbf{0}}(\pi')(x) \geq N$. We first prove that

Claim 62. *For every $M \in \mathbb{N}$ there is a run π'_b of $\mathcal{A}$ from q_a to q_b such that $\text{eval}_{\mathbf{0}}(\pi_a\pi'_b) = (M', 0)$ for some $M' > M$.*

First, by Claim 44 we have that $-CN \leq \text{eval}_{\mathbf{0}}(\pi_a)(x) \leq CN$. The claim is immediate if π_b contains a certificate of type A.1, A.2, or A.3. Indeed, we simply repeat the cycles that occur there a sufficient number of times. If π_b contains a certificate of type B, then $\pi_b = p_1 \xrightarrow{\pi_1}\!\!\!\!\!\rightarrow\; p_2 \xrightarrow{\pi_2}\!\!\!\!\!\rightarrow\; p_3 \twoheadrightarrow \cdots \twoheadrightarrow p_{n-1} \xrightarrow{\pi_{n-1}}\!\!\!\!\!\rightarrow\; p_n \xrightarrow{\pi_n}\!\!\!\!\!\rightarrow\; p_1$ is a cycle such that the overall effect on x is positive. Since every sub-path $\pi_1, \ldots, \pi_n$ contains a cycle with a positive effect on y and a non-negative effect on x, there is a path π''_b that repeats each such cycle $M + CN$ times. Note that π''_b is now a cycle with positive effect on x for any value of x smaller than $M + CN$. Thus, we can take π'_b to be the cycle π''_b taken $M + CN$ times.

This finishes the proof of the claim. Now, it suffices to show the following:

Claim 63. *For every $M \in \mathbb{N}$ there exists a run π'_c such that* $\mathrm{eval}_{\mathbf{0}}(\pi_a \pi'_b \pi'_c)(x) \geq M$.

We prove this claim by induction on the number of occurrences of η in π_c. Assume that π_c does not contain any occurrences of η. By Claim 62, there exists a path π'_b such that $\mathrm{eval}_{\mathbf{0}}(\pi_a \pi'_b)(x) \geq M + CN$. Thus, $\mathrm{eval}_{\mathbf{0}}(\pi_a \pi'_b \pi_c)(x) \geq M$ by Claim 44, since the length of π_c is bounded.

Assume now that there is at least one occurrence of η in π_c. Then π_c can be represented as follows:

$$\pi_c = q_b \xrightarrow{\rho_1,\alpha_1}\!\!\!\!\rightarrow\, q_1 \xrightarrow{\rho_2,\alpha_2}\!\!\!\!\rightarrow\, q_2 \xrightarrow{\rho_3,\alpha_3}\!\!\!\!\rightarrow \cdots \xrightarrow{\rho_r,\alpha_r}\!\!\!\!\rightarrow\, q_{\mathrm{fin}}\,,$$

where the cutting points are states reached after reading η. Assume that there is a run $q_{\mathrm{ini}} \xrightarrow{\pi_{i-1},w_{i-1}}\!\!\!\!\rightarrow\, q_{i-1}$ such that $\mathrm{eval}_{\mathbf{0}}(w_{i-1})(x) \geq M'$, for every $M' \in \mathbb{N}$. Then we can use the cycle with positive effect on y inside ρ_i in order to conclude that there exists a run $q_{\mathrm{ini}} \xrightarrow{\pi_i,w_i}\!\!\!\!\rightarrow\, q_i$ such that $\mathrm{eval}_{\mathbf{0}}(w_i) \geq M'$, for every $M' \in \mathbb{N}$. This concludes the proof. □

5.3.2 Unboundedness implies the existence of a witness

Lemma 64. *If $\mathcal{L}(\mathcal{A})$ is not bounded, there exists an unboundedness witness.*

Proof. Assume that $\mathcal{L}(\mathcal{A})$ is not bounded. Let $M \coloneqq 15C^2N^3$ and $W \coloneqq 12CN^2$. Let π be the shortest accepting run of $\mathcal{A}$ such that $\mathrm{eval}_{\mathbf{0}}(\pi)(x) > M + W$, and let w be the word labelling π. Since π is x-aligned, it can be split into two parts

$$\pi = q_{\mathrm{ini}} \xrightarrow{\pi_{\mathrm{pre}}}\!\!\!\!\rightarrow\, q_0 \xrightarrow{\pi_{\mathrm{suf}}}\!\!\!\!\rightarrow\, q_{\mathrm{fin}}$$

for some $q_0 \in Q$ and $q_{\mathrm{fin}} \in Q_{\mathrm{fin}}$, such that π_{suf} is the shortest suffix of π satisfying $\text{out-tree}(\pi) = \text{out-tree}(\pi_{\mathrm{suf}})$. Note that $\mathrm{eval}_{\mathbf{0}}(\pi_{\mathrm{pref}})(x) = 0$, and π_{suf} features no substitution ν that resets x (*i.e.*, for which $\nu(x) = 0$). Recall that we defined

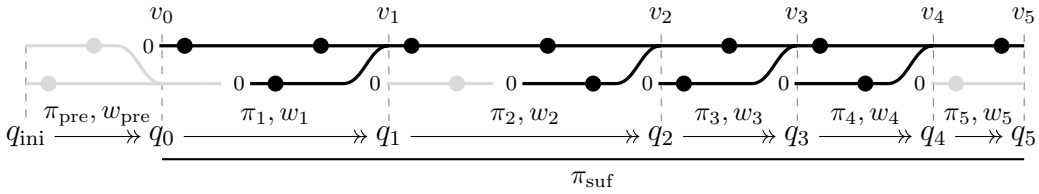
$$\eta = \{x \leftarrow \min\{x, y\},\ y \leftarrow 0\} \text{ and } \rho = \{x \leftarrow x,\ y \leftarrow 0\}\,.$$

Since π is x-aligned, we have that $w_{\mathrm{suf}} \in (A_X \cup \{\eta, \rho\})^*$, where w_{suf} is the word labelling π_{suf}. Let $n \in \mathbb{N}$ be the number of substitutions η in w. Split the run π_{suf} into segments (some possibly empty)

$$q_0 \xrightarrow{\pi_1,w_1}\!\!\!\!\rightarrow\, q_1 \xrightarrow{\pi_2,w_2}\!\!\!\!\rightarrow\, q_2 \twoheadrightarrow \cdots \twoheadrightarrow q_n \xrightarrow{\pi_{n+1},w_{n+1}}\!\!\!\!\rightarrow\, q_{n+1}$$

such that $q_1, \ldots, q_n$ are all the states reached directly after reading η each time. Define $v_i \coloneqq \mathrm{eval}_{\mathbf{0}}(\pi_{\mathrm{pref}} \pi_1 \cdots \pi_i)(x)$ for $0 \leq i \leq n+1$. Observe that $v_0 = 0$ and $v_{n+1} \geq M + W$.

Example 65. *Consider a run π partitioned into segments as defined above:*



Let us overlook the slight inaccuracy that a run reaching a large value would have many more additive transitions (cf. Claim 44). The out-tree(w) *is drawn in black, other (irrelevant) lines are drawn in light grey. Run π_{suf} is the shortest one that contains all black lines. There are $n = 4$ occurrences of η in w_{suf}, thus π_{suf} is split into 5 parts, and runs $\pi_1, \ldots, \pi_4$ end with a transition labelled by η.*

We first show that unboundedness guarantees the existence of a sustainable part π_c of a witness. The rest of the proof is done in three steps. First, we prove a claim that allows us to find a sustainable part π_c of a witness. With it, we can shift all our attention to the construction of a pumpable run. Next, we consider two cases, corresponding to two types of pumpable runs. Define $m \coloneqq \max\{i | v_i < M\}$.

Claim 66. *For every $i > m$ and every transition $s \xrightarrow{\nu} t$ in $\mathcal{A}$ such that t is a state visited by π_i and $\nu(y) = 0$, there exists a sustainable run π_c from t to q_{fin} of length at most $3N^2$.*

We prove the claim by downward induction on i, the index of the segment π_i incident with the state t of ν. Base case ($i = n+1$) is trivial, as $w_{n+1} \in A_X^*$. Assume our claim holds for $i+1$. Fix a transition $s \xrightarrow{\nu} t$ such that t is a state visited by π_i, and $\nu(y) = 0$. Similarly to the case of a trivial boundedness witness, a steep increase on y implies existence of a run $\pi_c' = t \xrightarrow{\alpha} r \xrightarrow{\theta, w_\theta} r \xrightarrow{\beta, w_\beta} s \xrightarrow{\eta} q_i$ such that $w_\theta, w_\beta \in A_X^*$, α, β are simple paths and θ a simple cycle, and that $\text{eff}(\theta)(y) > 0$. Finally, by applying the inductive hypothesis to $s \xrightarrow{\eta} q_i$, we get a pumpable π_c'' from q_i to q_{fin}; we have thus constructed a sustainable run $\pi_c'\pi_c''$, as required.

It remains to show how to find π_b, the pumpable part of the run. We consider two cases.

Case 1: $v_{i+1} - v_i > 4CN$ for some $i \in \mathbb{N} \cap [m, n]$. (aim: pumpable run of type A)
Fix such $i \in \mathbb{N}$. Let $\mathcal{A}' = (S, Q \cup \{q_{i+1}'\}, q_i, \{q_{i+1}'\}, \delta')$, where δ' is constructed from δ by removing transitions with minimum substitutions, and adding $q \xrightarrow{\eta} q_{i+1}'$ whenever $q \xrightarrow{\eta} q_{i+1}$ for some $q \in Q$. Note that $w_i \in \mathcal{L}(\mathcal{A}')$ and that $\text{eval}_{\mathbf{0}}(w_i)(x) > 4CN$, thus automaton $\mathcal{A}'$ satisfies the premises of Lemma 49. Using this lemma, we obtain a short pumpable run π_b' of type A of $\mathcal{A}'$ – cases A.1, A.2 and A.3 were designed to match all possible loop arrangements. It naturally induces π_b in $\mathcal{A}$ from q_i to q_{i+1}

of the same properties. Since q_i is reachable from q_{ini}, there exists a short run π_a between them. Finally, by Claim 66, we obtain a sustainable part π_c of the witness, which completes the analysis of this case.

Case 2: $v_{i+1} - v_i \leq 4CN$ for all $i \in \mathbb{N} \cap [m, n]$.(aim: witness exists or contradiction) This case proves to be more difficult, as it involves a more complicated type B pumpable run. The proof is by contradiction. Here, since $v_{n+1} - v_m > (M+W) - M = 12CN^2$ and each v_{i+1} provides an increase of at most $4CN$ compared to the previous value v_i, any maximal increasing subsequence of $(v_i)_{m \leq i \leq n}$ must have at least $\frac{12CN^2}{4CN} = 3N$ elements. Therefore, there exist $k, \ell \in \mathbb{N}$ such that

$$m \leq k < \ell \leq n, \quad v_k < v_\ell, \quad M < v_i < M + W \quad \text{for} \quad k \leq i \leq \ell, \quad \text{and} \quad q_k = q_\ell,$$

thus $\pi_{(k\ell]} \coloneqq \pi_{k+1} \cdots \pi_l$ is a cycle.

For each i, $k \leq i \leq l$, define $c_i \in \mathbb{K}$ to be the effect on x of the run π_i without its last transition labelled with η. We have $v_{i+1} \leq v_i + c_i$, and therefore $\sum_{i=k}^{\ell-1} c_i > v_\ell - v_k > 0$. Assume that $\mathcal{A}$ has no pumpable run of type B from q_k to q_ℓ (otherwise we obtain a witness easily). Therefore, there exists $i \in \mathbb{N} \cap [k, l]$ such that π_i decomposes into

$$\pi_i \coloneqq q_{i-1} \xrightarrow{\alpha,u}\!\!\!\!\rightarrow r_1 \xrightarrow{\beta,v}\!\!\!\!\rightarrow r_2 \xrightarrow{\eta} q_i,$$

where β is the run of maximal length labelled by some $v \in A_X^*$ without ρ, run α is labelled by $u \in (A_X \cup \{\rho\})^*$ (possibly empty), and no simple cycle θ with $\operatorname{eff}(\theta) \in \mathbb{N} \times \mathbb{N}_+$ is reachable from r_1 and backwards-reachable from r_2. Note that $\operatorname{eval}_{\mathbf{0}}(\pi_{\text{pre}}\pi_1\pi_2 \cdots \pi_{i-1}\alpha)(y) = 0$.

Assume that α contains a simple cycle θ. If $\operatorname{eff}(\theta)(x) \leq 0$, this cycle can be removed from π without decreasing the output value, which contradicts the assumption that π is the shortest. If otherwise $\operatorname{eff}(\theta)(x) > 0$, we obtain a pumpable run of type A.1 featuring θ and a simple path to r_1. Again, a sustainable run to the final state q_{fin} is guaranteed by Claim 66, and thus in this case the proof is finished.

Hence, we can assume that α does not have simple cycles. Due to Claim 44, we have that $\operatorname{eff}(\alpha)(x) \in [-CN, CN]$. By Lemma 48, $\operatorname{eff}(\beta)$ decomposes into $\operatorname{eff}(\beta) = \operatorname{eff}(\beta') + (A, B)$ for a run β' from r_1 to r_2 of length $\leq N$, and $(A, B) \in \operatorname{Cone}_{\mathbb{N}}(\Theta)$, where Θ is the set of simple cycles that are reachable from r_1 and backwards-reachable from r_2 by transitions not involving ρ. By assumption, $\operatorname{eff}(\theta) \notin \mathbb{N} \times \mathbb{N}_+$ for any $\theta \in \Theta$. Thus, for each $\theta \in \Theta$, either $\operatorname{eff}(\theta)(x) < 0$ or $\operatorname{eff}(\theta)(y) \leq 0$. Hence, for θ with $\operatorname{eff}(\theta)(y) > 0$, we have $\operatorname{eff}(\theta)(x) < 0$. Take $\theta_{\circlearrowleft}$ such that $\operatorname{eff}(\theta_{\circlearrowleft}) = (a, b)$, $b > 0$ (and hence $a < 0$) and $\frac{b}{-a}$ is the largest possible among cycles satisfying these conditions.

Every simple cycle has length at most N, therefore its effect belongs to $[-CN, CN]^2$. Thus, $\frac{CN}{1} \geq \frac{b}{-a}$. Let $\ell(t) \coloneqq \frac{b}{a}t$. If there exists θ' such that $\mathrm{eff}(\theta')$ lies above line ℓ, then we have identified two cycles that span a cone having a nonempty intersection with the positive quadrant; this yields a pumpable run of type A.3, and, by Claim 66, we get a sustainable run starting at q_i.

Otherwise, $\mathrm{Cone}_{\mathbb{N}}(\Theta)$ lies below ℓ. Since π_i ends with η and has effect at least M, $\mathrm{eff}(\beta)(y) > M$, therefore $B > 0$. This in turn implies $A < 0$, because (A, B) is bellow ℓ. Hence $B < \ell A = \frac{b}{a}A \leq -CNA$. We know that $\mathrm{eval}_{(v_{i-1},0)}(\pi_i) \geq M$, therefore

$$\begin{cases} \mathrm{eval}_{(v_{i-1},0)}(\alpha\beta)(x) \geq M \\ \mathrm{eval}_{(v_{i-1},0)}(\alpha\beta)(y) \geq M \end{cases} \quad \text{and thus} \quad \begin{cases} v_{i-1} + \mathrm{eff}(\alpha)(x) + \mathrm{eff}(\beta')(x) + A \geq M \\ 0 \qquad\qquad + \mathrm{eff}(\beta')(y) + B \geq M \end{cases}$$

Since $v_i < M + W$, and effects of simple runs α and β' are bounded by Claim 44, we get

$$\begin{cases} \cancel{M} + W + 2CN + A \geq \cancel{M} \\ CN + B \geq M \end{cases} \quad \text{and} \quad \begin{cases} 12C^2N^3 + 2C^2N^2 \geq -CNA \\ B \geq 15C^2N^3 - CN \end{cases}$$

Since $B < -CNA$, we have $15C^2N^3 - CN < 12C^2N^3 + 2C^2N^2$ and finally $3C^2N^3 < 2C^2N^2 + CN$ which yields a contradiction that concludes the proof. □

5.4 Output-minimum CRAs

In this section, we consider CRAs in which minimum substitutions can only appear in the output function. We call such CRAs *output-minimum* for brevity. The main results of this section are as follows.

Theorem 67. *Boundedness of output-minimum CRAs with no transpositions is coNP-complete, even if the numbers in the substitutions are presented in unary.*

Theorem 68. *Boundedness of output-minimum CRAs is PSPACE-complete, even if the numbers in the substitutions are presented in unary.*

For the rest of the section, fix the set of registers $X \coloneqq \{x_1, \ldots, x_d\}$. Let $\eta_{X'} \coloneqq \{x_1 \leftarrow \min\{x \mid x \in X'\}\}$ for $X' \subseteq X$. Once again, we use the formalism of regular languages of substitutions presented in Section 5.2. Recall that $\mathrm{Sub}_{\mathrm{elem}}(X) = T_X \cup R_X \cup A_X \cup M_X$. Since we are considering output-minimum CRAs, similarly to Claim 45, we can assume that the alphabet contains only elementary substitutions from $T_X \cup R_X \cup A_X$, with the only exception of a minimum transition which comes at

the end of the word. Thus, in Section 5.4.1 and Section 5.4.2 we consider the language boundedness problem for NFAs $\mathcal{A}$ such that for some $X' \subseteq X$, we have, respectively,

$$\mathcal{L}(\mathcal{A}) \subseteq (A_X \cup R_X)^* \eta_{X'} \text{ and}$$
$$\mathcal{L}(\mathcal{A}) \subseteq (A_X \cup R_X \cup T_X)^* \eta_{X'}.$$

As shown in Section 5.2, this is enough to prove Theorems 67 and 68.

5.4.1 Output-minimum CRAs with no transpositions

Proposition 69. *Boundedness for regular subsets of $(A_X \cup R_X)^* \eta_{X'}$ is coNP-complete, even if the numbers in the substitutions are presented in unary.*

Proof. As a certificate of unboundedness, we consider substitutions $\alpha_1, \ldots, \alpha_{d+1}$, $\beta_1, \ldots, \beta_d$ respecting the conditions of Lemma 49, together with a run π of $\mathcal{A}$ witnessing that $\alpha_1 \beta_1^+ \alpha_2 \ldots \alpha_d \beta_d^+ \alpha_{d+1} \eta_{X'} \subseteq \mathcal{L}(\mathcal{A})$. Checking the second and third conditions of Lemma 49 is trivial and by having π in the certificate, it is also easy to check the first condition in linear time.

Checking the last conditions requires a bit more work. As argued in the proof of Lemma 49, all substitutions in $\beta_1, \ldots, \beta_d$ can be modified so that they become additive substitutions without changing the value of $\mathrm{eval}_{\mathbf{0}}(\alpha_1 \beta_1^{a_1} \alpha_2 \ldots \alpha_d \beta_d^{a_d} \alpha_{d+1} \eta_{X'})(x_1)$. Now, let the vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d$ be the effects of the modified substitutions $\beta_1, \ldots, \beta_d$. By Theorem 46, we only need to solve the following linear program

$$\exists a_1, \ldots, a_d \in \mathbb{Q}_{\geq 0} \text{ s.t. } a_1 \boldsymbol{v}_1 + a_2 \boldsymbol{v}_2 + \cdots + a_d \boldsymbol{v}_d > 0,$$

which can be done in polynomial time.

To prove coNP-hardness, we reduce the satisfiability problem, which is NP-complete [42], to the complement of the boundedness problem.

Problem 70. SATISFIABILITY
Input: *A set $C = \{c_1, \ldots, c_m\}$ of clauses over boolean variables $p_1, \ldots, p_n$.*
Output: *Yes, if and only if there exists an assignment of Boolean values to the variables satisfying all the clauses.*

As registers, we take the set of all clauses: $X = C = \{c_1, \ldots, c_m\}$. Let $C_p :- \{c \in C | p \vDash c\}$ and $C_{\neg p} :- \{c \in C \mid \neg p \vDash c\}$ be the sets of clauses satisfied by $p = \top$ and $p = \bot$, respectively. Also, for a literal x, let $\mathrm{inc}(x) := \{c \leftarrow c + 1 | c \in C_x\}$. Consider the generalised NFA $\mathcal{A}$ in Figure 5.2.

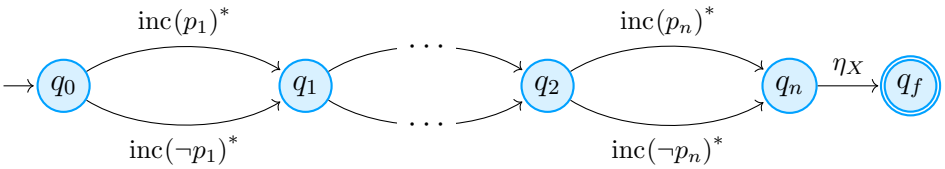


Figure 5.2: Generalised NFA $\mathcal{A}$ for satisfiability. Transitions are labelled by regular expressions.

It is readily seen that for any $N \in \mathbb{N}$, there exists a word

$$w \in \left(\mathrm{inc}(C_{p_1})^N \cup \mathrm{inc}(C_{\neg p_1})^N\right) \cdots \left(\mathrm{inc}(C_{p_n})^N \cup \mathrm{inc}(C_{\neg p_n})^N\right)\eta_X$$

such that $\mathrm{eval}_{\mathbf{0}}(w)(x_1) \geq N$ if and only if the variable assignment induced by w for variables $p_1, \ldots, p_n$ satisfies all the clauses $c_1, \ldots, c_m$. □

5.4.2 Output-minimum CRAs with transpositions

Proposition 71. *Boundedness for regular subsets of $(A_X \cup R_X \cup T_X)^*\eta_X$ is in PSPACE.*

The intuition for the proof is the following. We prove containment in PSPACE by operating on an exponentially larger NFA $\mathcal{P}_\mathcal{A}$, called permutation NFA. This NFA encodes all possible register permutations inside its state space, and hence its alphabet contains only additive and reset substitutions. It can be checked in NPSPACE whether this larger NFA admits a certificate of the type presented in Lemma 49. By Savitch's theorem we get that the problem is in PSPACE [85].

Proof. Fix a finite alphabet $S \subset A_X \cup R_X \cup T_X$ and let $C = \mathrm{maxc}(S)$. Fix NFA $\mathcal{A} = (Q, S \cup \{\eta_{X'}\}, \delta, q_{\mathrm{ini}}, Q_{\mathrm{fin}})$ such that $\mathcal{L}(\mathcal{A}) \subseteq S^*\eta_{X'}$. Let G_X be the set of all permutations of the set X and let $\mathcal{P}_\mathcal{A} = (Q', S' \cup \{\eta_{X'}\}, \delta', q'_{\mathrm{ini}}, Q'_{\mathrm{fin}})$, where $Q' = Q \times G_X$, $S' = S \setminus T_X$, and $Q'_{\mathrm{fin}} = Q_{\mathrm{fin}} \times G_X$. We also take ${q_{\mathrm{ini}}}' = (q_{\mathrm{ini}}, \tau_0)$, where τ_0 is the identity permutation. Let $\mathrm{id} \in S$ be the additive substitution that adds 0 to every register. For $s \in T_X$ and $\tau \in \mathcal{G}_X$ we define $(\tau \circ s)(x) = s(\tau(x))$. Finally, δ'contains transitions

$$\begin{aligned} ((q,\tau), \mathrm{id}, (q', \tau \circ s)) &\in \delta' \qquad \text{for every } (q, s, q') \in \delta \text{ such that } s \in A_X, \\ ((q,\tau), s, (q', \tau)) &\in \delta' \qquad \text{for every } (q, s, q') \in \delta \text{ such that } s \notin T_X. \end{aligned}$$

Clearly, $\mathcal{A}$ is unbounded if and only if $\mathcal{P}_\mathcal{A}$ is unbounded if and only if there exist words $\alpha_1, \ldots, \alpha_{d+1}, \beta_1, \ldots, \beta_d$ that adhere to the conditions of Lemma 49 and run π of $\mathcal{P}_\mathcal{A}$ witnessing that $\alpha_1\beta_1^+\alpha_2 \ldots \alpha_d\beta_d^+\alpha_{d+1}\eta_{X'} \subseteq \mathcal{L}\mathcal{P}_\mathcal{A}$. Since $|Q'| = |Q| \cdot d!$, we

can store one state with polynomial space. Hence, an NPSPACE algorithm can non-deterministically search for the run π without constructing $\mathcal{P}_\mathcal{A}$ explicitly. Verifying the first three conditions of Lemma 49 can be done on the fly in linear space. Also, a non-deterministic algorithm can guess and verify that $\boldsymbol{v}_1 \ldots, \boldsymbol{v}_d$ are the effects of cycles $\beta_1, \ldots, \beta_d$ on the fly. Finally, it is easy to verify that a positive linear combination of them has positive effect on all registers. Thus, we can conclude the argument by recalling that NPSPACE=PSPACE by Savitch's theorem [85]. □

Proposition 72. *Boundedness for regular subsets of* $(A_X \cup R_X \cup T_X)^* \eta_X$ *is PSPACE-hard, even if the numbers in the substitutions are presented in unary.*

Again, we start by giving the intuition for the proof. We reduce from the DFA intersection problem, which is PSPACE-complete [57]. For every state of each DFA, we create a separate register in the constructed CRA $\mathcal{C}$. We simulate reading a letter by all the DFAs by moving a large value to the registers corresponding to the new active states of the DFAs, and keeping the values of all remaining registers zero. These large values come from a self-loop transition in the initial state of $\mathcal{C}$, and $\mathcal{C}$ cannot return to the initial state afterwards. All DFAs accept the same word if and only the large values can be simultaneously brought to the registers corresponding to the final states of the DFAs. The output of $\mathcal{C}$ is thus set to be the minimum of these registers.

Proof. We reduce from the following PSPACE-complete problem [57]:

Problem 73. DFA INTERSECTION
Input: $n \in \mathbb{N}$, *alphabet* Σ, n *DFAs* $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, q_{\text{ini}}^{(i)}, Q_{\text{fin}}^{(i)})$, $1 \leq i \leq n$.
Output: *Yes, if and only if there exist a word accepted by all the DFAs.*

We can assume that each DFA has only one final state, which can be ensured as follows: add a new letter # to Σ and two new states q_i^+, q_i^- to each $\mathcal{A}_i$. For each i, make this new letter # send all states from $Q_{\text{fin}}^{(i)}$ to q_i^+, and all other states of $\mathcal{A}_i$ to q_i^-. Make the new letter induce a self-loop for both q_i^+, q_i^- and make q_i^+ to be the only final state in each DFA.

We construct an NFA $\mathcal{A}$ such that the language $\mathcal{L}(\mathcal{A})$ is bounded if and only if $\bigcap_{1 \leq i \leq n} \mathcal{L}(\mathcal{A}_i)$ is empty. Let $X = \left\{ q_i^{(j)} \mid 1 \leq j \leq n, q_i \in Q_j \right\}$. The idea is that the registers correspond to the states of the DFAs, and register $r_i^{(j)}$ has a positive value after reading $w \in \Sigma$ if and only if the i'th state in $\mathcal{A}_j$ is active after reading w. Next, we describe $\mathcal{L}(\mathcal{A})$ in terms of a regular language.

Let $\nu_{\text{inc}} = \left\{ q_{\text{ini}}^{(i)} \leftarrow q_{\text{ini}}^{(i)} + 1 \mid 1 \leq i \leq n \right\}$ be a substitution that increments the registers representing initial states for each $\mathcal{A}_i$. Also, for every $\sigma \in \Sigma$, let

$$T_{i,\sigma} = \{\{q' \leftarrow q\} \cup \{q \leftarrow 0 \mid q \neq q'\} \mid (q, \sigma, q') \in \delta_i, q \neq q'\} \text{ and}$$
$$T_\sigma = T_{1,\sigma} \cdot T_{2,\sigma} \cdot \ldots \cdot T_{n,\sigma},$$
$$T = \bigcup_{\sigma \in \Sigma} T_\sigma.$$

There is a substitution in $T_{i,\sigma}$ that simulates every transition inside $\mathcal{A}_i$ for letter $\sigma \in \Sigma$. The idea is that after we guess the next letter $\sigma \in \Sigma$, for each $\mathcal{A}_i$ we need to simulate the transition that is executed when reading this letter. If we pick the correct transition, we move our positive value from register q_i to q_i', and resetting all other registers does not change their values. However, if we pick a wrong transition, we reset our positive value and we can never recover. Then, a substitution from T_σ simulates executing a transition labelled by letter σ in all $\mathcal{A}_i$ and a substitution from T simulates choosing a letter $\sigma \in \Sigma$ and executing a transition labelled by σ in all $\mathcal{A}_i$. Finally let $\nu_{\text{out}} = \left\{ x \leftarrow \min\{q_{\text{fin}}^{(i)} \mid 1 \leq i \leq n\} \right\}$. We argue that $\mathcal{L}(\mathcal{A}) = \nu_{\text{inc}}^* \cdot T^* \cdot \nu_{\text{out}}$ is unbounded if and only if $\bigcap_{1 \leq i \leq n} \mathcal{L}(A_i)$ is non-empty.

Consider a word $w = \sigma_1 \ldots \sigma_m \in \Sigma^*$ and an integer $N \in \mathbb{N}$. For every $1 \leq i \leq n$ it follows inductively that there exists a word $w_j' \in \text{inc}^{N+1} \cdot T_{\sigma_1} \cdot T_{\sigma_2} \cdot \ldots \cdot T_{\sigma_j}$ such that $\text{eval}_{\mathbf{0}}(w_j')(q^{(i)}) = N + 1$, for $q \in Q_i$ if and only if $q_{\text{ini}}^{(i)} \xrightarrow{\sigma_1 \ldots \sigma_j} q^{(i)}$. Thus, there exists a word $w_m' \in \nu_{\text{inc}}^* \cdot T^* \cdot \nu_{\text{out}}$ such that $\text{eval}_{\mathbf{0}}(w_m') = N + 1$ if and only if $\bigcap_{1 \leq i \leq n} L(A_i)$ is non-empty. □

5.5 Stateless CRAs

In this section, for every $d \geq 2$ we present a fairly restricted family of unbounded CRAs with d registers such that the length of a shortest run outputting a value $N \in \mathbb{N}$ is lower bounded by $F_{d-1}(N)$, the $(d-1)$st function in the hierarchy of fast-growing functions. These functions are defined as follows. Let $F_1(n) = 2n$ and for every $k \geq 2$ let $F_k(n) = F_{k-1} \circ \ldots \circ F_{k-1}(1)$, where $\circ$ denotes the composition of functions, and this composition is taken n times. For example, $F_2(n) = 2^n$ and $F_3(n) = 2^{2^{\cdot^{\cdot^{\cdot^2}}}} = \text{Tower}(n)$, where exponentiation is taken n times. We construct these CRAs inductively in the following theorem.

Theorem 74. *For every $d \geq 2$, there exists a stateless CRA $\mathcal{C}$ with d registers and d transitions such that for every $N \in \mathbb{N}_+$, any run of $\mathcal{C}_d$ that outputs a value of at least N must have length at least $F_{d-1}(N)$.*

Proof. For $d = 2$ and $d = 3$ consider the two CRAs in Figure 5.3.

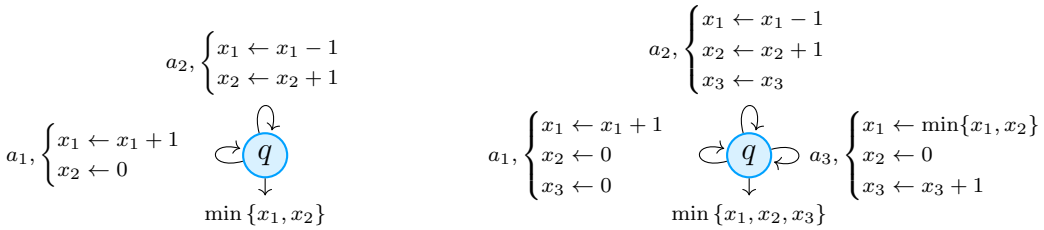


Figure 5.3: Unbounded CRAs $\mathcal{C}_2$ (left) and $\mathcal{C}_3$ (right) with 2 and 3 registers.

Let us prove that $\mathcal{C}_2$ and $\mathcal{C}_3$ satisfies the statement of the lemma. Let $N \in \mathbb{N}_+$ be an arbitrary positive integer. Since we are dealing with stateless CRAs, we denote a configuration $\langle q, \{x_1 \leftarrow k_1, x_2 \leftarrow k_2, \ldots, x_n \leftarrow k_n\}\rangle$ by a vector $\langle k_1, k_2, \ldots, k_n\rangle$. Clearly, for both $\mathcal{C}_2$ and $\mathcal{C}_3$, the transitions labelled by a_1 are the only ones that can increase the value of register x_1 and since these transitions reset the values of all other counters, any run outputting N must start by taking this transition m many times, $m > 0$, reaching in $\mathcal{C}_2$ and $\mathcal{C}_3$ the configurations $\langle m, 0\rangle$ and $\langle m, 0, 0\rangle$, respectively. The value m can be seen as the initial budget that is necessary for increasing the values of other registers. Clearly, the shortest run that outputs N in $\mathcal{C}_2$ reaches the configuration $\langle 2N, 0\rangle$, then takes N times the transition labelled by a_2 and outputs. Since it must start by getting to the configuration $\langle 2N, 0\rangle$, its length is at most $F_1(N) = 2N$.

Let now π_3 be a shortest run in $\mathcal{C}_3$ that outputs N. Clearly, π_3 needs to increase the value of register x_3. The transition labelled by a_3 is the only one that increases x_3, however, it contains a minimum update for x_1. Since π_3 is a shortest path outputting N, before reading a_3 it reaches a configuration in which the values of registers x_2, x_3 are equal, otherwise some transitions can be removed from it without changing the output value.

Thus, π_3 initialises the budget by reading a_1^m, and then, before reading a_3, it reads a word in a_2^* which applies the function $F_1^{-1}(\cdot) = \frac{\cdot}{2}$ to the value of register x_1. We argue that in order to output N, $m = F_2^{-1}(N) = 2^N$. Indeed, π_3 must reach a value m in register x_1 and then apply N many times the function $F_1^{-1}(\cdot)$ to register x_1, so $m = F_2^{-1}(N)$. Thus, the length of π_3 must be longer than $F_2(N)$. Furthermore, we see that π_3 has the following shape $\langle 0, 0, 0\rangle \twoheadrightarrow \langle F_2(N), 0, 0\rangle \twoheadrightarrow \langle N, N, N\rangle$.

Assume now that there exists $\mathcal{C}_{d-1}$ with the property from the statement of the lemma. We modify it by adding a new letter a_d to the alphabet Σ, and extend the substitutions of the transitions as follows:

- add $x_d \leftarrow 0$ to a_1,

- add $x_d \leftarrow x_d$ to a_i for $1 < i < d-1$, and
- let a_d be $\{x_1 \leftarrow \min\{x_1, \ldots, x_{d-1}\}, x_2 \leftarrow 0, \ldots, x_{d-1} \leftarrow 0, x_d \leftarrow x_d + 1\}$.

We know that there exists a shortest path π_{d-1} with the following shape $\langle 0, \ldots, 0\rangle \twoheadrightarrow \langle F_{d-2}(N), 0, \ldots, 0\rangle \twoheadrightarrow \langle N, N, \ldots, N, 0\rangle$. So, in order to increase the register x_d by one, we need to have enough budget on register x_1 to be able to apply the function F_{d-2}^{-1} to its value. Since we need to increase the value of x_d by one N times, it follows that we need to repeat this process N times so that π_d has shape $\langle 0, \ldots, 0\rangle \twoheadrightarrow \langle F_{d-1}(N), 0, \ldots, 0\rangle \twoheadrightarrow \langle N, N, \ldots, N, N\rangle$. Thus its length must be at least $F_{d-1}(N)$. □

5.6 Discussion

The most obvious open problem left by this work is the decidability of boundedness for copyless linear CRAs with resets with more than two registers. We conjecture that it is decidable for arbitrary number of registers. Our techniques and the shapes of the witnesses for the two-register case might be useful for proving that. The crucial aspect seems to be the notion that a register can support another one when taking iterated minimums. Thus, one might attempt to create a hierarchy of registers such that one has an unboundedness witness for a subset of registers before attempting to increase an extra register.

As stated in the Preliminaries, we cannot model the behaviour of CRAs with Presburger arithmetic dues to the presence of minimum operations in the update functions. This makes the computed functions highly non-linear as they can compute iterated minimums. However, it would be interesting to connect the subclasses of CRAs with decidable properties to a suitable extension of Presburger arithmetic in order to find new decidable extension as we do in Chapter 3.

Another interesting open problem is the precise complexity of the two-register case where numbers in the substitutions are presented in binary. We have proved that this problem is NL-hard and in coNP, but no better bound is known even if minimum substitutions are only allowed in the output. In the latter case, it follows from our results that the witness of unboundedness consists of at most two cycles and some paths connecting them. This relates to the following natural problem whose complexity we were not able to find in the literature. Let $G = (V, E)$ be a digraph, and $\omega : E \to \mathbb{Z}^2$ be a (bi-criteria) weighting function on its edges. Given G and ω, find a cycle in G such that the sum of weights of its edges is component-wise positive.

This is of course a generalisation of the problem of finding a cycle of negative weight in a digraph, which can be solved in polynomial time by e.g. Bellman-Ford-Moore algorithm [9].

Chapter 6

Conclusions

The general motif of this thesis is model of computation with counters. In both Chapter 3 and Chapter 5 we developed new pumping lemmas for the relevant models. The process of removing the loops one by one from a path and understanding their effect proved crucial for proving these lemmas and we believe that the process can be applied to many other models of computation with counters. Also, the connection between arithmetic theories and models of computation proved to be a lot more general and we managed to extend the concept of automatic structures. We hope that this concept can be extended even further, so that new results both in the area of arithmetic theories and formal models can be obtained by leveraging this connection.

Bibliography

[1] Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2009.

[2] Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? *Information and Computation*, 282:104651, 2022.

[3] Shaull Almagor, Michaël Cadilhac, Filip Mazowiecki, and Guillermo A. Pérez. Weak cost register automata are still powerful. In Mizuho Hoshi and Shinnosuke Seki, editors, *Developments in Language Theory*, pages 83–95, Cham, 2018. Springer International Publishing.

[4] Shaull Almagor, Michaël Cadilhac, Filip Mazowiecki, and Guillermo A. Pérez. Weak cost register automata are still powerful. *International Journal of Foundations of Computer Science*, 31(6):689–709, 2020.

[5] Rajeev Alur, Loris D'Antoni, Jyotirmoy Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 13–22, 2013.

[6] Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium*, volume 7966 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2013.

[7] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms*, 6(2):28:1–28:36, 2010.

[8] Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.

[9] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

[10] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

[11] Ben-Ari and Mordechai. *Mathematical Logic for Computer Science*. Springer London, third edition, 2012.

[12] Michael Benedikt, Dmitry Chistikov, and Alessio Mansutti. The Complexity of Presburger Arithmetic with Power or Powers. In *50th International Colloquium on Automata, Languages, and Programming*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 112:1–112:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[13] Murphy Berzish, Joel D. Day, Vijay Ganesh, Mitja Kulczynski, Florin Manea, Federico Mora, and Dirk Nowotka. Towards more efficient methods for solving regular-expression heavy string constraints. *Theor. Comput. Sci.*, 943:50–72, 2023.

[14] Alexis Bès. A survey of arithmetical definability. 2013.

[15] Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is pspace-complete. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 32–43. IEEE Computer Society, 2015.

[16] Achim Blumensath and Erich Grädel. Automatic structures. In *Logic in Computer Science, LICS*, pages 51–62. IEEE Computer Society, 2000.

[17] V Bruyere. *Entiers et automates finis, mémoire de fin d'études*. PhD thesis, Master's thesis, University of Mons, Belgium, 1985.

[18] Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p-recognizable sets of integers. *Bulletin of the Belgian Mathematical Society Simon Stevin*, 1(2):191–238, 1994.

[19] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

[20] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Transactions on Computational Logic*, 11(4):23:1–23:38, 2010.

[21] Gregory Cherlin and Françoise Point. On extensions of Presburger arithmetic. In *Proc. 4th Easter Model Theory conference, Gross Köris*, pages 17–34, 1986.

[22] Dmitry Chistikov, Rupak Majumdar, and Philipp Schepper. Subcubic certificates for CFL reachability. *Proceedings of the ACM on Programming Languages*, 6(POPL), 2022.

[23] David C. Cooper. Theorem proving in arithmetic without multiplication. *Mach. Intell*, 7:91–99, 1972.

[24] Karel Culík and Jarkko Kari. Digital images and formal languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 599–616. Springer, 1997.

[25] Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. *Journal of the ACM*, 68(1):1–28, 2020.

[26] Wojciech Czerwiński, Engel Lefaucheux, Filip Mazowiecki, David Purser, and Markus A. Whiteland. The boundedness and zero isolation problems for weighted automata over nonnegative rationals. In Christel Baier and Dana Fisman, editors, *37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 15:1–15:13. ACM, 2022.

[27] Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *Annual Symposium on Foundations of Computer Science, FOCS*, pages 1229–1240. IEEE, 2021.

[28] Wojciech Czerwinski and Lukasz Orlikowski. Lower bounds for the reachability problem in fixed dimensional VASSes. In *Symposium on Logic in Computer Science, LICS*, New York, NY, USA, 2022. Association for Computing Machinery.

[29] Laure Daviaud. Containment and equivalence of weighted automata: Probabilistic and max-plus cases. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications - 14th International Conference*, volume 12038 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2020.

[30] Laure Daviaud. Register complexity and determinisation of max-plus automata. *ACM SIGLOG News*, 7(2):4–14, 2020.

[31] Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. A generalised twinning property for minimisation of cost register automata. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 857–866, 2016.

[32] Laure Daviaud and Andrew Ryzhikov. Universality and forall-exactness of cost register automata with few registers. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science*, volume 272 of *LIPIcs*, pages 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[33] Andrei Draghici, Christoph Haase, and Florin Manea. Semënov Arithmetic, Affine {VASS}, and String Constraints. In *41st International Symposium on Theoretical Aspects of Computer Science*, volume 289 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[34] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1-2):69–86, 2007.

[35] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. In *Handbook of weighted automata*, pages 175–211. Springer, 2009.

[36] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer Berlin, Heidelberg, 1st edition, 2009.

[37] Manfred Droste and Dietrich Kuske. Weighted automata. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 113–150. European Mathematical Society Publishing House, Zürich, Switzerland, 2021.

[38] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Annual Symposium on Foundations of Computer Science, FOCS*, pages 143–152, 2010.

[39] Henri-Alex Esbelin and Malika More. Rudimentary relations and primitive recursion: A toolbox. *Theoretical Computer Science*, 193(1):129–148, 1998.

[40] James Evans. *The History and Practice of Ancient Astronomy*. Oxford University Press, 1998.

[41] Patrick C. Fischer, Albert R. Meyer, and Arnold L Rosenberg. Counter machines and counter languages. *Mathematical systems theory*, 2:265–283, 1968.

[42] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[43] Sheila A. Greibach. The hardest context-free language. *SIAM Journal on Computing*, 2(4):304–310, 1973.

[44] Sheila A. Greibach. Remarks on the complexity of nondeterministic counter languages. *Theoretical Computer Science*, 1(4):269–288, 1976.

[45] Florent Guépin, Christoph Haase, and James Worrell. On the existential theories of Büchi arithmetic and linear p-adic fields. In *Proc. Symposium on Logic in Computer Science, LICS*, pages 1–10, 2019.

[46] Christoph Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018.

[47] Kosaburo Hashiguchi. New upper bounds to the limitedness of distance automata. *Theor. Comput. Sci.*, 233(1-2):19–32, 2000.

[48] Kosaburo Hashiguchi, Kenichi Ishiguro, and Shuji Jimbo. Decisability of the equivalence problem for finitely ambiguous automata. *International Journal of Algebra and Computation*, 12(03):445–461, 2002.

[49] Philipp Hieronymi and Christian Schulz. A strong version of cobham's theorem. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 1172–1179, New York, NY, USA, 2022. Association for Computing Machinery.

[50] John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159, 1979.

[51] Ismaël Jecker. 22.1 complexity of fixed vas reachability. https://automata.exchange/22.01-complexity-fixed-vas-reachability/, 2022. Accessed: 11.12.2024.

[52] Toghrul Karimov, Florian Luca, Joris Nieuwveld, Joël Ouaknine, and James Worrell. On the decidability of presburger arithmetic expanded with powers. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2755–2778. Association for Computing Machinery, 2025.

[53] Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In *Logical and Computational Complexity, LCC*, volume 960 of *Lect. Notes Comp. Sci.*, pages 367–392. Springer, 1995.

[54] Daniel Kirsten and Sylvain Lombardy. Deciding Unambiguity and Sequentiality of Polynomially Ambiguous Min-Plus Automata. In *26th International Symposium on Theoretical Aspects of Computer Science*, volume 3 of *LIPIcs*, pages 589–600, 2009.

[55] Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004.

[56] S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Symposium on Theory of Computing, STOC*, pages 267–281. ACM, 1982.

[57] Dexter Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 254–266. IEEE Computer Society, 1977.

[58] Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–426, 1994.

[59] Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *Annual Symposium on Foundations of Computer Science, FOCS*, pages 1241–1252. IEEE, 2021.

[60] Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *Symposium on Logic in Computer Science, LICS*, pages 56–67. IEEE Computer Society, 2015.

[61] Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2019.

[62] Hing Leung and Viktor Podolskiy. The limitedness problem on distance automata: Hashiguchi's method revisited. *Theor. Comput. Sci.*, 310(1-3):147–158, 2004.

[63] Richard J. Lipton. *The reachability problem requires exponential space*. Research report (Yale University. Department of Computer Science). Department of Computer Science, Yale University, 1976.

[64] Yuri Matiyasevich. Enumerable sets are diophantine. In *Soviet Mathematics - Doklady*, volume 11, pages 354–358, 1970.

[65] Ernst W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13*, pages 238–246. ACM, 1981.

[66] Ernst W. Mayr. An algorithm for the general petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.

[67] Filip Mazowiecki and Cristian Riveros. Maximal partition logic: Towards a logical characterization of copyless cost register automata. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic*, volume 41 of *LIPIcs*, pages 144–159. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[68] Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. *Journal of Computer and System Sciences*, 100:1–29, 2019.

[69] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, USA, 1967.

[70] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[71] Danny Nguyen and Igor Pak. Short Presburger arithmetic is hard. *SIAM J. Comput.*, 51(2):17:1–30, 2022.

[72] Derek C. Oppen. A 222pn upper bound on the complexity of presburger arithmetic. *Journal of Computer and System Sciences*, 16(3):323–332, 1978.

[73] Françoise Point. On the expansion of $(\mathbb{N}, +, 2^x)$ of Presburger arithmetic. Unpublished manuscript, 2010.

[74] Mojzesz Presburger. *Über die Vollständigkeiteines gewissen Systems der Arithmetik ganzer Zahlen, in welchen die Addition als einzige Operation hervortritt.* Comptes-Rendus du ler Congres des Mathematiciens des Pays Slavs, 1929.

[75] Julien Reichert. *Reachability games with counters: decidability and algorithms.* PhD thesis, École normale supérieure de Cachan-ENS Cachan, 2015.

[76] Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *Journal of Computer and System Sciences*, 32(1):105–135, 1986.

[77] Wojciech Rytter. A hardest language recognized by two-way nondeterministic pushdown automata. *Information Processing Letters*, 13(4):145–146, 1981.

[78] Uwe Schöning. *Logic for Computer Scientists.* Birkhäuser, Boston, MA, 1989.

[79] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., USA, 1986.

[80] Rich Schroeppel. *A two counter machine cannot calculate* 2^N. Artificial Intelligence Memo 257. Massachusetts Institute of Technology, 1972.

[81] Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2):245–270, 1961.

[82] Nicole Schweikardt. Arithmetic, first-order logic, and counting quantifiers. *ACM Transactions on Computational Logic (TOCL)*, 6(3):634–671, 2005.

[83] A. L. Semënov. On certain extensions of the arithmetic of addition of natural numbers. *Mathematics of the USSR-Izvestiya*, 15(2):401, 1980.

[84] Imre Simon. On semigroups of matrices over the tropical semiring. *RAIRO Theor. Informatics Appl.*, 28(3-4):277–294, 1994.

[85] Michael Sipser. *Introduction to the Theory of Computation.* Course Technology, Boston, MA, third edition, 2013.

[86] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Symposium on Theory of Computing, STOC*, pages 137–146, New York, NY, USA, 1982. Association for Computing Machinery.

[87] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science (FOCS 1985)*, pages 327–338, 1985.

[88] Roger Villemaire. The theory of (n,+,vk,vl) is undecidable. *Theor. Comput. Sci.*, 106(2):337–349, 1992.

[89] Andreas Weber. Finite-valued distance automata. *Theoretical Computer Science*, 134(1):225–251, 1994.

[90] V. Weispfenning. The complexity of almost linear diophantine problems. *Journal of Symbolic Computation*, 10(5):395–403, 1990.

[91] Volker Weispfenning. The complexity of almost linear diophantine problems. *Journal of Symbolic Computation*, 10(5):395–403, 1990.

[92] Pierre Wolper and Bernard Boigelot. On the construction of automata from linear arithmetic constraints. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 1–19, 2000.