Algebraic Abstraction in Formal Methods
(形式手法における代数的抽象化)

by

Takamasa Okudono
奥殿 貴仁

Dissertation

submitted to the Department of Informatics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
博士(情報学)

総合研究大学院大学
The Graduate University for Advanced Studies, SOKENDAI
March 2021

Acknowledgements

I would like to express my deepest gratitude to my supervisor Ichiro Hasuo. He invited me to join the Ph.D. course and gave me the opportunity to pursue my research. He shared the joy of my discoveries with me, and guided me with great patience when my research was not going well. I would also like to thank the thesis committee members Yasuhiko Minamide, Katsumi Inoue, Mahito Sugiyama, Taro Sekiyama for their comments and discussions.

I would like to thank all my collaborators. First, I would like to thank Andy King of the University of Kent. He and I met at APLAS 2017 held in Suzhou and discussed Craig's interpolants, and then he invited me to Canterbury. The research in Chapter 4 of this thesis was completed based on the research there. In this research, he provided me a good problem, listened carefully to my ideas, taught me the basics of performance tuning, and taught me research techniques. He also took me to great restaurants in Canterbury, which were my favorite memories of the UK. I would like to thank Elena Gutiérrez, Masaki Waga, Yuki Nishida, Kensuke Kojima, Kohei Suenaga, Kengo Kido for working with me on my research. I would also like to thank Rafael Wisniewski, Hayato Waki, for accepting my research visit. I would also like to thank the anonymous reviewers for their interesting comments on our submitted papers.

I thank Jérémy Dubut for discussing with me and reading my thesis. He quickly understood my vague research ideas and gave me advice based on his solid understanding of mathematics. He read my draft of this thesis carefully and gave me many comments.

I would like to thank ERATO MMSD project. Without the support of the office, I would have missed many critical deadlines. I would also like to thank Ikuo Ishimura of SOKENDAI's mental health counseling team for his help and support during my difficult time.

Finally, I express my gratitude to my family and friends for various supports.

Takamasa Okudono / 奥殿 貴仁

Tokyo, March 2021

Abstract

Formal methods are development techniques to describe and analyze systems and specifications mathematically and help to ensure the safety of systems. As computer programs are being used in safety-critical systems, formal methods are becoming increasingly important. Model checking is a style of formal methods. For a system described as an automaton and a specification for the system, model checking constructs a proof of safety in an algorithmic manner. By transforming a program into an automaton in a proper way, model checking can be applied to verifying program, and program verification by model checking is realized. As model checking automatically constructs safety proofs, it needs less expertise in formal methods to use than other formal methods, but it has problems with applicability to complex systems and scalability.

This thesis aims to expand the range of problems that model checking can be efficiently applied to by looking at algebraic structures underlying target systems. We expect that it improves the applicability of formal methods and encourages the industry to use model checking. There are multiple benefits of using the properties of algebraic structures in model checking. Firstly, writing a target system as an algebraic system leads to the algebraic description of the abstract states, i.e., the subsets of the memory states of the systems. Algebraic descriptions of the abstract states allow us to manipulate them so that we can check the properties of the system computationally. Secondly, as many algebraic structures satisfy the associative law, if a system is compatible with an algebraic structure, the exhaustive search of the system can be done efficiently with memoization or parallelization. Thirdly, we researchers can get an insight by inspecting the underlying structure of a target system and connecting it with other mathematical objects.

In Chapters 3 and 4, we develop methods of interpolant generation, which can be regarded as abstraction methods in program verification. Interpolants (or Craig intepolants) are considered to be essential predicates to analyze the properties of programs, and commonly used in model checking of various types of programs. Chapter 3 aims to improve interpolant generation for programs with polynomials. Dai et al.'s technique could not generate interpolants when the given data are "barely disjoint." We make the technique works in this situation by (1) proposing a substructure of the polynomial ring $\mathbb{R}[\vec{X}]$, called the strict cone, and by (2) proposing a simplification of ratios to deal with numerical erros that occurs in interpolant generation. Chapter 4 aims to improve interpolant generation in the bit-vector theory for programs in which integer variables cause the overflow and wraparound. For this task, Griggio's

technique uses an interpolant generation in the linear integer arithmetic, and it has an advantage that it generates interpolants while preserving the semantics of the program. However, they reported that it sometimes fails at generating interpolants by failing to deal with the overflow. We look at the group $(\mathbb{Z}/n\mathbb{Z})^d$, which represents the memory space, combine it with a torus, and propose a new technique, called *boxing and gapping*, for this task.

Chapters 5 and 6 aim to approximate a complex system with a weighted finite automaton (WFA). A WFA is a quantitative extension of a deterministic finite automaton, and models a function from words to real numbers. Applying model checking to complex systems is often impractical. We advocate that the scenario of applying a formal method to an approximated system to certify the safety of the approximated system is useful to know the partial safety of the original system. Chapter 5 aims to approximate a recurrent neural network with a weighted finite automaton over $\mathbb{R}$. For this task, we propose a method to compare a candidate approximated WFA with an recurrent neural network to be approximated. The approximation is driven by Balle and Mohri's algorithm, which learns a weighted finite automata from given data, and the comparing method is used as a subroutine of Balle and Mohri's algorithm. Chapter 6 aims to extend Balle and Mohri's algorithm for general semirings, including the max-plus semiring, and improve the flexibility of techniques to approximate complex systems with weighted finite automata. We show that the naive extension of Balle and Mohri's algorithm outputs an "unfaithful" weighted finite automaton, which ignores some given data, as some properties of fields do not necessarily hold in general semirings. We prove that *column-closedness* is necessary to ensure "faithfulness," and propose a new algorithm to assure the column-closedness.

Table of Contents

Tables

Figures

Chapter 1

Introduction

This thesis aims to broaden the range of systems that we can efficiently verify by inspecting the algebraic structures found in the target class of the systems. We can then discover verification techniques by inspecting the algebraic structures, observing the structure's property, and applying appropriate algorithms corresponding to the structure.

1.1 Program Verification and Model Checking

Assuring the safety of a program is an important problem, as many industrial products are controlled by computers and softwares. Such products include automotive control systems [66, 137, 138], flight assistance [68], artificial pancreas [23], and pacemakers [5]. The flaws in such systems can seriously damage people's property, health, and lives.

Formal methods are techniques to describe and analyze systems and specifications in a mathematical manner. A class of such, called *formal verification* (or *verification*), certifies that a system is safe in any situation if there is a proof that the system satisfies a specification describing the safety of the system. There are two main types of formal verification: *deductive verification* and *model checking*.

Deductive verification is a method to certify the safety property of a system by describing the system as a definition and the property as a theorem, and proving that the system satisfies the property in a formal logic. We usually use proof assistants like Coq [15], Isabelle [100], or Lean [40] for deductive verification. The pro is its flexibility: any system and specification described in a formal logic can be targets. The con is that describing systems and specifications in the formal logic needs many man-months and a high level of expertise. Though automated theorem proving technologies like Sledgehammer [108] partially automate to write proofs, the end-to-end automation for complex systems is not realistic yet. CompCert [83], which aims to a formally verified C compiler, and the finding of a vulnerability in OpenSSL [75] are examples

that deductive verification works well.

On the other hand, model checking takes a state machine as a system and a specification for the system, and checks whether the system satisfies the specification in an algorithmic manner. In other words, it generates a proof of safety if the system is safe, and returns a counterexample if the system is unsafe. The pro is its facility: once a system and a specification are given, a model checking algorithm automatically finds a proof, or a counterexample. Its user does not have to do anything. The cons are *the inapplicability to complex systems* and *the lack of scalability*:

- *Inapplicability to complex systems*: As the operations supported by a target system become richer or a specification becomes more complex, the complexity of the predicates describing abstract states also increases during model checking. Since the cost of model checking is dominated by the manipulation of abstract states, the complexity of systems leads to the difficulty of model checking.
- *Lack of scalability*: Since model checking examines all the execution paths of a target system, the time and space to run model checking explode as the system becomes larger.

One of the most successful model checking techniques is for ω-automata and LTL specifications [130], and is supported by popular model checkers, such as SPIN [63] and NuSMV [31]. Various types of model checkers are researched and developed for various purposes and systems: SLAM [10] for the safety checking of device drivers, CBMC [79] for programs written in C language, Java PathFinder [131] to inspect concurrent programs written in Java, and MoCHi [77] for higher-order programs.

Model checking is being accepted by the software industry. Amazon reported that they found some bugs in Amazon Web Service using TLA+ and TLC model checker [81, 98, 99]. SQUARE ENIX (a video game company in Japan) reported that they applied TLA+ and TLC model checker to check the validity of the database of *FINAL FANTASY XV POCKET EDITION* [64].

1.2 Abstraction

To inspect more complex systems with model checking, we need *abstraction*, which creates a simpler system that inherits the properties of an original system by combining multiple states of the system under consideration as one state. By applying an appropriate abstraction, the complexity problem is solved by transforming transitions as simple transitions on some kinds of automata or labeled transition systems, and the size problem is solved by combining states. For example, let's think about the following state machine:

- The states are $(x, v) \in \mathbb{R}^2$,
- the transition is replacing the current state (x, v) with $(x + v, v + 0.01)$, and
- the initial state is $(x, v) = (0, 0)$.

The run looks like

$$\to (0,0) \to (0,0.01) \to (0.01,0.02) \to (0.03,0.03) \to (0.06,0.04) \to (0.1,0.05) \to \ldots.$$

We want to prove that the state x is always nonnegative. Of course, as the state machine has infinite states $\mathbb{R}^2$, it is impossible to conclude the safety by checking all the states and the transitions. An useful abstraction here is to separate the state space into four abstract states $A_1 = \{\, (x, v) \mid x \geq 0, v \geq 0 \,\}$, $A_2 = \{\, (x, v) \mid x < 0, v \geq 0 \,\}$, $A_3 = \{\, (x, v) \mid x \geq 0, v < 0 \,\}$, and $A_4 = \{\, (x, v) \mid x < 0, v < 0 \,\}$. As the initial state $(0, 0)$ belongs to A_1, A_1 is the initial abstract state. Any state (x, v) in A_1 goes to A_1 again by applying the transition, which can be solved by SMT solvers supporting the linear arithmetic, and we know that the run on the abstract state machine is

$$\to A_1 \to A_1 \to A_1 \to \ldots.$$

This concludes that our goal "x is always nonnegative" is proved. Finding good predicates, "$x \geq 0$ and $v \geq 0$" in the above example, is important in the abstraction. *Craig interpolants* play an important role in finding good predicates for it in Chapters 3 and 4.

1.3 Algebraic Abstraction

The aim of this thesis is to make the abstraction of a target system by looking at the algebraic structure[*1] under the system, and broaden the range of systems that can be verified.

The philosophy underlying this thesis is

> Look at an algebraic system whose underlying set corresponds to the state space of the system and whose operations describe the transition of the system. It leads us to find an efficient verification method.

Each chapter in this thesis is a practice of this philosophy.

There are three advantages of using algebraic structures for abstraction.

- In model checking, an abstract state is a subset of the memory space defined by a proposition. By restricting the operations we use to describe abstract states and transitions, we can make it easy to compute the image of an abstract state through a function and

[*1] Refer to Section 1.3.6 for the discussion about "algebraic structures."

to compute the relationship between two abstract states. If the operations are derived from an algebraic structure, we can use this structure for those calculations. The theorems make it easier to manipulate the abstract states, and thus make model checking easier.

Popular structures are vector spaces on a field (typically $\mathbb{R}$ or $\mathbb{Q}$). Programs with linear operations can be covered with the analysis using vector spaces. An abstract state in this situation is represented with polytopes, and they can be manipulated with various algorithms like the computation of convex hulls, linear programming, or Fourier-Motzkin elimination [35]. Using vector spaces is also effective in interpolation techniques [116].

- Many algebraic structures satisfy the associative law, and in a computational perspective, this means that operations can be done in any order. This allows us to memoize and reuse some of the computations, or parallelize the computations to inspect the property of a target system. Gutiérrez [56] used the associativity of matrix multiplications, and invstigated the properties of weighted finite automata.
- As major algebraic structures are investigated well by mathematicians, we can research the target system described by an algebraic structure with plenty of intuition by connecting the target with other mathematical objects.

The target systems, algebraic structures, and properties of the algebraic systems used in this thesis are shown in Table 1.1, and let's see the role of them in the following sections.

1.3.1 Algebraic Structure in Chapter 3

In Chapter 3 and its previous work [39], we focus on the ring $\mathbb{R}[\vec{X}]$ for the verification of programs containing polynomials. Deriving from the fact that the real field $\mathbb{R}$ is an ordered field, $\mathbb{R}[\vec{X}]$ has a substructure called *cone*. A theorem called *Positivstellensatz* holds, which converts a problem to check the emptiness of a shape defined by polynomials and inequalities into a problem to find appropriate polynomials in cones, ideals, and multiplicative monoids. Conditions written with cones, ideals, and multiplicative monoids can be written as an optimization problem called *sum-of-squares optimization*, and it is finally reduced to *semidefinite programming*, which can be solved efficiently. The previous work proposed a way to construct an optimization problem to compute an interpolant, which is an overapproximation of an abstract state, using this theorem as a hint.

Our algebraic contribution is to propose a new substructure in $\mathbb{R}[\vec{X}]$, called the *strict cone*, replace the multiplicative monoid with a strict cone, and propose a new way to compute an interpolant.

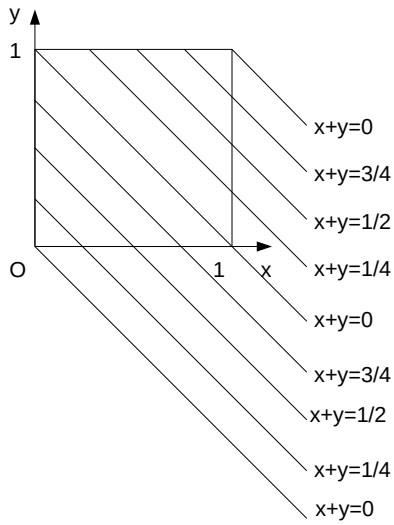
Table 1.1: Relationship of target systems, algebraic structures, and properties of the structures used in this thesis

Chap.	system	algebraic structure	property of the structure
3	programs with polynomials	ring ($\mathbb{R}[\vec{X}]$)	Positivstellensatz is applicable. Substructures of $\mathbb{R}[\vec{X}]$ are compatible with sum-of-squares constraints.
4	programs with the overflow	group ($(\mathbb{Z}/n\mathbb{Z})^d$)	Direct products of $\mathbb{Z}/n\mathbb{Z}$ form tori.
5	RNN	monoid (Σ^*), vector space over $\mathbb{R}$ ($\mathbb{R}^{\Sigma^*}$)	Balle and Mohri's algorithm is applicable to WFA over $\mathbb{R}$. Systems of linear equations over $\mathbb{R}$ are efficiently solvable.
6	(potentially) RNN	monoid (Σ^*), the max-plus semiring ($\mathbb{R}_{\max}$), semimodule over the max-plus semiring ($\mathbb{R}_{\max}^{\Sigma^*}$)	Systems of linear equations over the max-plus semiring are efficiently solvable.

1.3.2 Algebraic Structure in Chapter 4

Chapter 4 targets a program in which an integer variable overflows, which means the wraparound occurs and the number following the maximum value is zero. To verify such a program, calculating an interpolant in the theory of Linear Integer Arithmetic is important. A useful algebraic structure to express the memory space is $(\mathbb{Z}/n\mathbb{Z})^d$ assuming that the number of variables is d and the maximum value of a variable is $n-1$.

As we shall see later, it is important to investigate shapes in the memory space $(\mathbb{Z}/n\mathbb{Z})^d$ that are defined by a linear inequality. For simplicity, let's think the case $d = 2$ and the constraint is the inequality $ax + by = c$, where $a, b, c \in \mathbb{Z}$. The set $(\mathbb{Z}/n\mathbb{Z})^2$ can be regarded as a discrete version of the torus $S_1^2 \simeq \mathbb{R}^2/\mathbb{Z}^2$. It is known that a straight line with a rational slope forms a closed curve in the torus (see Section 15 of [128]). Using this fact as a hint, we know that the family of straight lines $ax + by = c$ with parameter c forms a "stratum" in $(\mathbb{Z}/n\mathbb{Z})^2$ (see Figure 1.1). This observation is the idea of *gapping* in Chapter 4, and we could come up with this by writing the system as an algebraic structure and expand the mathematical association.



3	3	0	1	2
2	2	3	0	1
1	1	2	3	0
0	0	1	2	3
y/x	0	1	2	3

Figure 1.1: Family of lines $x + y = c$ with parameters $c = 0, 1/4, 1/2, 3/4$ on the torus $\mathbb{R}^2/\mathbb{Z}^2$ (Left). Values $x + y$ on $(\mathbb{Z}/4\mathbb{Z})^2$ (Right). Note that the same line appears many times on the left picture because of the quotient by $\mathbb{Z}^2$.

1.3.3 Algebraic Structure in Chapter 5

Chapter 5 aims at approximating a target RNN by a *weighted finite automata (WFA)* over $\mathbb{R}$. A weighted finite automaton is an automaton whose memory space is a finite-dimensional vector space $\mathbb{R}^d$ and whose transitions are endomorphisms in the vector space.

For WFAs on $\mathbb{R}$, Balle and Mohri algorithm [12] is known, and it dynamically learns a target system and generates a WFA on $\mathbb{R}$.

As the WFA inference proceeds by matrix products and since matrix multiplication is associative, it can reuse the partial calculations by memoization and parallelization. This is in contrast to an RNN inference, which requires sequential processing of input.

1.3.4 Algebraic Structure in Chapter 6

Chapter 6 aims at extending Balle and Mohri's algorithm, which is used in Chapter 5, for general semirings including the max-plus semiring. The max-plus semiring is defined as follows (the precise definition is given in Chapter 6). Its underlying set is $\mathbb{R} \cup \{ -\infty \}$, where $-\infty$ is a symbol. The zero (the identity element of addition) is $-\infty$ and the one (the identity element

of multiplication) is $0 \in \mathbb{R}$. The addition and multiplication are defined by

$$x \oplus y = \begin{cases} \max_{\mathbb{R}}(x, y) & (x, y \in \mathbb{R}) \\ x & (y = -\infty) \\ y & \text{otherwise} \end{cases}, \tag{1.1}$$

$$x \otimes y = \begin{cases} x +_{\mathbb{R}} y & (x, y \in \mathbb{R}) \\ -\infty & \text{otherwise} \end{cases}. \tag{1.2}$$

Semirings have strange properties compared to fields, and these are obstacles to the extension of Balle and Mohri's algorithm. For example, for d-tuples in a semiring $\mathbb{S}$ $v_1, \ldots, v_n, v' \in \mathbb{S}^d$, even if $v_1, \ldots, v_n$ are "linearly independent" and v' cannot be expressed as a "linear combination" of $v_1, \ldots, v_n$, v_1 could be expressed as a combination of $v_2, \ldots, v_n, v'$. This phenomenon cannot happen in fields because of the subtraction, and causes a problem for the extension.

1.3.5 Algebraic Abstraction: Examples and Non-examples

In this thesis, abstraction refers to the transformation of a mathematically-described system into a simple and verifiable system by combining multiple states together. Abstractions where algebraic theorems can be used in the process, or where algebraic theorems can be used to analyze the transformed system are called "algebraic abstractions." To make the meaning of "algebraic" clear, we are giving some examples of what we call "algebraic" abstraction and what we do not call "algebraic" abstraction in this paper.

Examples of "algebraic" theorems or algorithms for abstraction are below:

- Simplex method / Fourier-Motzkin algorithm (in the vector space $\mathbb{R}^n$): For a system whose state space is expressed as the vector space $\mathbb{R}^n$, a technique to combine multiple states into one abstract state by linear inequalities is often used. An abstract state made in this manner is a region defined by the conjunction of linear inequalities, which is a polytope. The algorithms like simplex method or Fourier-Motzkin algorithm enables operations on polyhedra (projection, simplification, and so on), and they are used for program verification [35].
- Farkas' lemma / Motzkin's theorem (in the vector space $\mathbb{R}^n$): Rybalchenko proposed an interpolation method in linear arithmetic using linear optimization, and applied it to program verification. To fit the interpolation problem into a linear optimization problem, they used Motkin's theorem (a variant of Farkas' lemma), which enables us to transform "less than ($<$)" conditions into "less than or equal to ($\leq$)" conditions.
- Positivstellensatz (in the polynomial ring $\mathbb{R}[\vec{X}]$): For a system whose state space is ex-

pressed as the Euclidean space $\mathbb{R}^n$, there are abstraction techniques to combine multiple states into one abstract state by polynomial inequalities. An abstract state made in this manner is a region defined by polynomial inequalities, which is called a semialgebraic set. We can use *real algebraic geometry* to investigate semialgebraic sets. Dai et al. proposed a method to calculate interpolants of the system of real inequalities based on Positivstellensatz, which is a theorem of real algebraic geometry [39].

- Gröbner basis / Buchberger's algorithm (in the polynomial ring $\mathbb{R}[\vec{X}]$): Sankaranarayanan et al. proposed a method to investigate a program using Buchberger's algorithm, which is an algorithm of computational algebraic geometry [117]. Their method tries to find a loop invariant, an invariant of the variables, described as algebraic varieties, shapes written with polynomial equalities. Gröbner basis is a normal expression of algebraic varieties, and Buchberger's algorithm enables us to compute Gröbner basis.
- Bezout's identity (in the integer ring $\mathbb{Z}$): Griggio's work [53] worked on finding interpolants in Linear Integer Arithmetic, a logic supporting linear inequalities and moduli. The algebraic property of the integer ring $\mathbb{Z}$ is used as follows. Interpolants of formulae A and B are typically calculated from the proof that $A \wedge B$ is unsatisfiable. If A and B are systems of linear equalities, we have to construct the proof that the system of linear inequalities $A \wedge B$ has no solution. This problem is reduced to calculating the GCD of the coefficients according to Bezout's identity, which is a theorem of elementary number theory.

Examples of "non-algebraic" abstraction are below:

- Discretization for continuous systems: For a system whose state space is expressed as an Euclidean space and whose transition is expressed as a differential equation, there is a technique to split the state space into meshes so that the abstract system is a finite LTS (explained in [118]). The transition of the abstract system is constructed by following the differential equation of the original system. After the abstraction, the verification of the original system is reduced to the verification of the finite LTS. As the construction of the transition does not use any theorems of algebraic structures and the analysis of the abstract system is straightforward, we do not regard this technique as algebraic abstraction.
- Partial order reduction: For a parallel system, there is a verification technique called *partial order reduction*. It searches the target parallel system in depth-first manner, bunches up execution paths whose results are the same, and construct a smaller abstract system than the target system. As both the constructed abstract system and the target

parallel system are just an LTSs, theorems of algebraic structures are not used to verify the abstract system. The point of the process of the abstraction is to find paths whose results are the same, and usually theorems of algebraic structures are not used. Hence we do not regard partial order reduction as algebraic abstraction. See [9,33] for details.

1.3.6 Algebraic Structure as Universal Algebra

In this section, we divert to the topic about a definition of "algebraic structure." Readers can safely skip over this section.

One of the most standard definitions of "algebraic structure" is in the context of *universal algebra*. An algebraic structure in universal algebra consists of a *signature* Σ and *equational formulae* E. A signature $\Sigma = (\Sigma_n)_{n\in\mathbb{N}}$, where each Σ_n is a set of symbols, defines the operations in the algebraic structure, and a set of equational formulae E defines the axioms, i.e., the relationships of the signatures required to hold in the algebraic structure. A pair (Σ, E) is called *algebraic specification*. A set X with n-arity functions $X^n \to X$ for every symbol of Σ_n respecting E is called a (Σ, E)-*algebra* or a *model* of the algebraic structure (Σ, E). The set X (forgetting the structure) is called the *underlying set*. Most of the algebraic structures used in this thesis are described in universal algebra. For example, the signature Σ for *monoids* is defined by $\Sigma_0 = \{ e \}$ (unit), $\Sigma_2 = \{ \cdot \}$ (multiplication), and $\Sigma_n = \emptyset$ for $n \notin \{ 0, 2 \}$. The equational formulae E are $e \cdot x = x \cdot e = x$ and $(x \cdot y) \cdot z = x \cdot (y \cdot z)$. For another example, The signature for Σ of *semimodules* over a semiring $\mathbb{S}$ is defined by $\Sigma_0 = \{ 0 \}$ (zero), $\Sigma_1 = \{ (c\cdot) \mid c \in \mathbb{S} \}$ (scalars), $\Sigma_2 = \{ + \}$ (addition), $\Sigma_n =$ for $n \geq 3$. The equational formulae E are (1) $0 + x = x$, (2) $(x + y) + z = x + (y + z)$, (3) $x + y = y + x$, (4) $c \cdot (x + y) = c \cdot x + c \cdot y$ for $c \in \mathbb{S}$, (5) $(c \oplus c') \cdot x = c \cdot x + c' \cdot x$ for $c, c' \in \mathbb{S}$, (6) $(c \otimes c') \cdot x = c \cdot (c' \cdot x)$ for $c, c' \in \mathbb{S}$, (7) $1_{\mathbb{S}} \cdot x = x$, (8) $0_{\mathbb{S}} \cdot x = 0$, and (9) $c \cdot 0 = 0$ for all $c \in \mathbb{S}$. As we saw in this example, a signature does not have to be finite.

1.4 Overview of the Contributions

First two chapters (Chapters 3 and 4) are about *interpolation*, which is a technique of abstraction in model checking.

1.4.1 Overview of Chapter 3

Interpolation of jointly infeasible predicates plays important roles in various program verification techniques such as invariant synthesis and CEGAR. Intrigued by the recent result by

Dai et al. that combines real algebraic geometry and SDP optimization in synthesis of polynomial interpolants, this chapter contributes its enhancement that yields *sharper* and *simpler* interpolants. The enhancement is made possible by: theoretical observations in real algebraic geometry; and our continued fraction-based algorithm that rounds off (potentially erroneous) numerical solutions of SDP solvers. Experiment results support our tool's effectiveness; we also demonstrate the benefit of sharp and simple interpolants on program verification examples.

1.4.2 Overview of Chapter 4

Much of an interpolation engine for bit-vector (BV) arithmetic can be constructed by observing that BV arithmetic can be modeled with linear integer arithmetic (LIA). Two BV formulae can thus be translated into two LIA formulae and then an interpolation engine for LIA is used to derive an interpolant, albeit one expressed in LIA. The construction is completed by back-translating the LIA interpolant into a BV formula whose models coincide with those of the LIA interpolant. This chapter develops a back-translation algorithm showing, for the first time, how back-translation can be universally applied, whatever the LIA interpolants. This avoids the need for deriving a BV interpolant by bit-blasting the BV formulae, as a backup process when back-translation fails. The new back-translation process relies on a novel geometric technique, called gapping, the correctness and practicality of which are demonstrated.

Next two chapters (Chapter 5 and Chapter 6) are about approximation of complex systems with automata learning methods.

1.4.3 Overview of Chapter 5

We present a method to extract a weighted finite automaton (WFA) from a recurrent neural network (RNN). Our algorithm is based on the WFA learning algorithm by Balle and Mohri, which is in turn an extension of Angluin's classic L*algorithm. Our technical novelty is in the use of *regression* methods for the so-called equivalence queries, thus exploiting the internal state space of an RNN to prioritize counterexample candidates. This way we achieve a quantitative/weighted extension of the recent work by Weiss, Goldberg and Yahav that extracts DFAs. We experimentally evaluate the accuracy, expressivity and efficiency of the extracted WFAs.

1.4.4 Overview of Chapter 6

Active learning of finite automata has been vigorously pursued for the purposes of analysis and explanation of black-box systems. In this chapter, we study an L^*-style learning algorithm for weighted automata over the *max-plus semiring*. The max-plus setting exposes a "consistency" issue in the previously studied semiring-generic extension of L^*: we show that it can fail to maintain consistency of tables, and can thus make equivalence queries on obviously wrong hypothesis automata. We present a theoretical fix by a mathematically clean notion of *column-closedness*. Our algorithm is applicable to general semirings.

1.5 Organization of the Thesis

Chapters 3–6 present our contributions we have looked over in Section 1.4. For each, we conclude the chapter and discuss the future work. Chapter 7 concludes the thesis.

Chapter 2

Preliminaries

In this chapter, we will provide some background knowledge of the general concepts that appear throughout this thesis. First, we describe automata learning, a technique for interactively inferring a regular language from a limited access to it. Next, we will explain the interpolation of logical expressions, which is commonly used in program verification, and its specific applications to program verification.

2.1 Automata Learning

2.1.1 Angluin's L* Algorithm

2.1.1.1 Problem Formulation

In this section, we describe Angluin's L* algorithm, which is the basis of automata learning. Angluin's L* algorithm is an algorithm for identifying black box regular languages. The algorithm uses limited access to the regular language, namely through called *membership queries* and *equivalence queries*, and raises them multiple times to identify the regular language. Henceforth, Angluin's L* algorithm will be referred to simply as the L* algorithm.

Mathematically, the L* algorithm takes as input an oracle answering the membership query $m\colon \Sigma^* \to \{\,0,1\,\}$ and an oracle answering the equivalence query $e : \{\,\mathrm{DFAs}\,\} \to \{\,\texttt{Equivalent}\,\} \sqcup \Sigma^*$ and outputs a minimum DFA A. The relationship between the two input oracles m, e and the output minimum DFA A is as follows:

> Let $L \subset \Sigma^*$ be a regular language. If m satisfies
>
> $$m(w) = \begin{cases} \texttt{tt} & ; w \in L \\ \texttt{ff} & ; w \notin L \end{cases} \quad \text{for } w \in \Sigma^*, \tag{2.1}$$

and e satisfies

$$e(A') = \begin{cases} \texttt{Equivalent} & ; L(A') = L \\ w & ; w \in L(A') \triangle L \end{cases} \quad \text{for a DFA } A' \tag{2.2}$$

then the algorithm terminates, and $L = L(A)$ holds[*1].

Intuitively, the above property means that if m determines whether a given word w belongs to the regular language L, and e determines whether the language recognized by a given DFA A' is equal to L, then the algorithm correctly find a DFA A that recognizes L.

The pair (m, e) is called a *teacher*, and the entity running the algorithm is called a *learner*. This is because the teacher is capable of answering queries about the language L, and the entity raising the queries seems to be trying to learn the language from the information provided by the teacher.

The word returned by e is called a *counterexample*, because it shows the difference bewteen the learner's guess and the teacher's answer. The learner uses the counterexample to improve the guess.

2.1.1.2 Example

An execution of the L* algorithm is shown using an example[*2]. Let the alphabet be $\Sigma = \{ a, b \}$ and the learned language L be

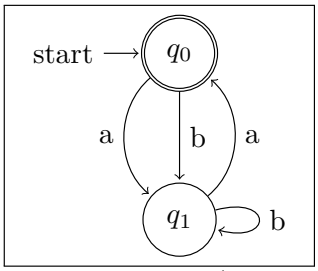
$$L = \{ w \in \Sigma^* \mid \text{Both } a \text{ and } b \text{ appear an even number of times in } w \}. \tag{2.3}$$

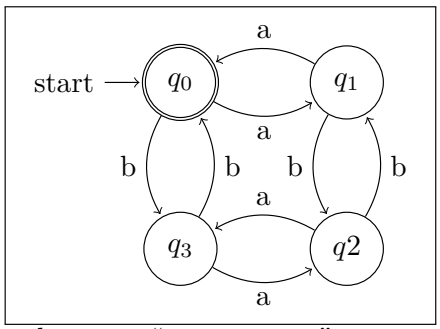
We use the language L to define m and e to create a teacher. Learning by the L* algorithm in this language L leads to the following access to the teacher (m, e). We represent this as a dialogue between Learner and Answerer.

1. Learner queries "Is 'ϵ' in L?" Teacher answers "Yes".
2. Learner queries "Is 'a' in L?" Teacher answers "No".
3. Learner queries "Is 'b' in L?" Teacher answers "No".
4. Learner queries "Is 'aa' in L?" Teacher answers "Yes".
5. Learner queries "Is 'ab' in L?" Teacher answers "No".

[*1] For sets S_1 and S_2, $S_1 \triangle S_2$ represents the symmetric difference of S_1 and S_2, which is $(S_1 \setminus S_2) \cup (S_2 \setminus S_1)$

[*2] Our demo to see the process of L* algorithm for an arbitrary language is provided online: `https://colab.research.google.com/drive/1zdFtXE1UpVJKmEFIWzaPh5ow8PS7Idf4`

6. Learner queries “Does 

recognize L? ” Teacher answers “No”, and returns a counterexample “aba $\in L$(this autom.) $\setminus L$”.
7. Learner queries “Is ‘abb’ in L?” Teacher answers “No”.
8. Learner queries “Is ‘abaa’ in L?” Teacher answers “No”.
9. Learner queries “Is ‘abab’ in L?” Teacher answers “Yes”.
10. Learner queries “Is ‘ba’ in L?” Teacher answers “No”.
11. Learner queries “Is ‘aaa’ in L?” Teacher answers “No”.
12. Learner queries “Is ‘abba’ in L?” Teacher answers “Yes”.
13. Learner queries “Is ‘abaaa’ in L?” Teacher answers “No”.
14. Learner queries “Is ‘ababa’ in L?” Teacher answers “No”.
15. Learner queries “Is ‘bb’ in L?” Teacher answers “Yes”.
16. Learner queries “Is ‘bba’ in L?” Teacher answers “No”.
17. Learner queries “Is ‘aab’ in L?” Teacher answers “No”.
18. Learner queries “Is ‘aaba’ in L?” Teacher answers “No”.
19. Learner queries “Is ‘abbb’ in L?” Teacher answers “No”.
20. Learner queries “Is ‘abbba’ in L?” Teacher answers “No”.
21. Learner queries “Is ‘abaab’ in L?” Teacher answers “No”.
22. Learner queries “Is ‘abaaba’ in L?” Teacher answers “Yes”.
23. Learner queries “Is ‘ababb’ in L?” Teacher answers “No”.
24. Learner queries “Is ‘ababba’ in L?” Teacher answers “No”.
25. Learner queries “Does 

recognize L? ” Teacher answers “Yes”, and returns the atom “`Equivalent`”.

Table 2.1: Hankel matrix of $L = \{ w \in \Sigma^* \mid \text{Both } a \text{ and } b \text{ appear an even number of times in } w \}$

prefix\suffix	ϵ	a	b	ab	$\cdots$
ϵ	tt	ff	ff	ff	$\cdots$
a	ff	tt	ff	ff	$\cdots$
aa	tt	ff	ff	ff	$\cdots$
ab	ff	ff	ff	tt	$\cdots$
aba	ff	ff	tt	ff	$\cdots$
abb	ff	tt	ff	ff	$\cdots$
abaa	ff	ff	ff	tt	$\cdots$
abab	tt	ff	ff	ff	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

2.1.1.3 Principle

■Constructing an automaton from a Hankel matrix The *Hankel matrix* of a language $L \subseteq \Sigma^*$ is a table of inifinite rows and infinite columns labeled by Σ^* whose cell at the p-th row and s-th column indicates whether $p \cdot s$ belongs to L or not. For example, the hankel matrix of the above language $L = \{ w \in \Sigma^* \mid \text{Both } a \text{ and } b \text{ appear an even number of times in } w \}$ looks like Table 2.1.

The r-th row of the Hankel matrix of L, where r is a word, contains the complete information of the (Brzozowski) derivative by r, which is $[r]_L \in \Sigma^*/\sim_L$[*3]: the c-th column of the r-th row is tt if and only if $c \in [r]_L$. Since the Myhill-Nerode theorem states that a language L is regular if and only if the quotient set $\Sigma^*/\sim_L$ is a finite set, the language L is regular if and only if there are only a finite number of different row contents in the Hankel matrix of L. By following the proof of the Myhill-Nerode theorem, we can construct the minimal DFA recognizing L as follow:

1. Let the states Q be the row contents in the Hankel matrix of L, which is identified with $\{ [w]_L \mid w \in \Sigma^* \}$. The size $\#Q$ is finite if L is regular.
2. For each $[w]_L \in Q$ and $\sigma \in \Sigma$, which is identified with the w-th row, the transition $\delta([w]_L, \sigma)$ is defined by the $w\sigma$-th row, which is identified with $[w\sigma]_L \in Q$. The well-definedness follows from the definition of $\sim_L$.

[*3] Recall that the relation is defined by $w \sim_L w' :\iff (ww'' \in L \iff w'w'' \in L)$ for words $w, w', w'' \in \Sigma^*$.

3. Let the ϵ-th row, which is identified with $[\epsilon]_L \in Q$, be the initial state q_0.
4. Let the final states be the rows whose ϵ-th columns are $\mathtt{tt}$, which correspond to the elements $[w]_L \in Q$ such that $\epsilon \in [w]_L$.

For example, we can construct the automaton in the above example from Table 2.1 as follows.

1. The table consists of four types of rows: $(\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots)$, $(\mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \dots)$, $(\mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \dots)$, and $(\mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \dots)$, so Q is defined by
$$Q = \{\, (\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots), (\mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \dots), (\mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \dots), (\mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \dots) \,\}. \tag{2.4}$$
2. We are finding where to go from the state $(\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots) \in Q$ by reading "a". Words whose rows are $(\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots)$ are ϵ, "aa", "abab", and so on, so we pick ϵ as a representative. As the concatination of the representative ϵ and the input character "a" is "a", and the row content of the "a"-th row is $(\mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \dots)$, we find the transition is $(\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots) \xrightarrow{a} (\mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \dots)$. By repeating this procedure for each $a, b \in \Sigma$ and element in Q, we find that
$$(\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots) \xrightarrow{a} (\mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \dots), (\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots) \xrightarrow{b} (\mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \dots) \tag{2.5}$$
$$(\mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \dots) \xrightarrow{a} (\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots), (\mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \dots) \xrightarrow{b} (\mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \dots) \tag{2.6}$$
$$(\mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \dots) \xrightarrow{a} (\mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \dots), (\mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \dots) \xrightarrow{b} (\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots) \tag{2.7}$$
$$(\mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \dots) \xrightarrow{a} (\mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \dots), (\mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \mathtt{tt}, \dots) \xrightarrow{b} (\mathtt{ff}, \mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \dots). \tag{2.8}$$
3. The initial state q_0 is $q_0 = (\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots) \in Q$, which is the content of the ϵ-th row.
4. The set of final states $F \subset Q$ is:
$$F = \{\, q \in Q \mid \text{the } \epsilon\text{-th element of } q \text{ is } \mathtt{tt} \,\} = \{\, (\mathtt{tt}, \mathtt{ff}, \mathtt{ff}, \mathtt{ff}, \dots) \,\}. \tag{2.9}$$

■Hankel Submatrix / Observation Table A Hankel matrix has an infinite amount of information and cannot be handled by a computer. Therefore, we need to consider its finite approximation. A *Hankel submatrix* of a language $L \subset \Sigma^*$ is a table of row labels P and column labels S with the p-th row and the s-th column indicating whether $p \cdot s$ belongs to L or not, where P is a finite prefix-closed set of words and S is a finite suffix-closed set of words. We call this table a Hankel submatrix for consistency with later discussions, but in the context of the L* algorithm, it is usually referred to as an *observation table*.

If we try to apply the same procedure above to the observation table, instead of the Hankel matrix, the following problems can occur:

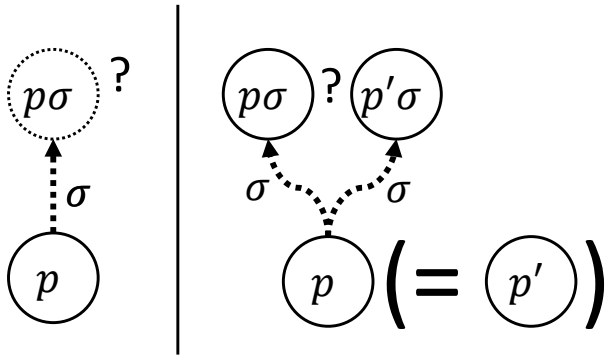


Figure 2.1: The problem of unclosedness (left) and the problem of inconsistency (right)

Unclosedness There can exist a row label $p \in P$ and a character $\sigma \in \Sigma$ such that the row content $\lambda s.\chi(ps \in L)$[*4] does not appear in the observation table, where s is a word in the column label set S. From the automata viewpoint, this problem means there is nowhere to go from the state corresponding to p reading σ. An observation table is said to be *closed* when there is no unclosedness problem in the observation table. See Figure 2.1 (left).

Inconsistency There can exist row labels $p, p' \in P$ whose corresponding row contents are the same $\lambda s.ps = \lambda s.p's$ and a character $\sigma \in \Sigma$ such that the row contents corresponding to the row labels $p\sigma$ and $p'\sigma$ are different $\lambda s.p\sigma s \neq \lambda s.p'\sigma s$. From the automata viewpoint, this problem means there are two different states to go from the state corresponding to p, which is the same as the state corresponding to p', reading σ. An observation table is said to be *consistent* when there is no inconsistency problem in the observation table. See Figure 2.1 (right).

Conversely, if an observation table is closed and consistent, then the same procedure above constructs an automaton that is compatible with the content of the observation table.

■Algorithm The basic idea of the L^* algorithm is to start with $P = \{\epsilon\}$ and $S = \{\epsilon\}$ and extend the observation table so that it is closed and consistent. Raising queries only when it is necessary to make the observation table closed and consistent reduces the number of queries.

To resolve the unclosedness, for $p \in P$ and $\sigma \in \Sigma$ making the observation table unclosed, we add $p\sigma$ to the row label set P, and raise membership queries to fill the unknown cells in

[*4] Here χ is the characteristic function. For a predicate $P(x)$, $\chi(x)$ is `tt` if $P(x)$ holds, and is `ff` if $P(x)$ does not hold.

the $p\sigma$-th row.

To resolve the inconsistency, for $p, p' \in P$ and $\sigma \in \Sigma$ making the observation table inconsistent, we add $\{ \sigma s \mid s \in S \}$ to the column label set S, and raise membership queries to fill the unknwon cells in the new columns. The new columns makes the row contents corresponding p and p' different because of the property of p, p' and σ, and resolves the inconsistency.

Wrapping up the above discussion yields the L^* algorithm below:

1. Let the row label set $P = \{ \epsilon \}$ and the column label set $S = \{ \epsilon \}$, and fill the observation table by raising membership queries.
2. Check the closedness of the observation table determined by (P, S). If it is not closed, update P by the procedure above.
3. Check the concistency of the observation table determined by (P, S). If it is not consistent, update S by the procedure above.
4. Repeat Step 2 and 3 until the update of P and S stops. When the update stops, the observation table is closed and consistent.
5. Construct a DFA from the closed and consistent observation table.
6. Raise an equivalence query by the constructed DFA.
7. If the answer is `Equivalent`, the learning is done. If the answer is a counterexample $w \in \Sigma^*$, add all the prefixes of w to P, and go back to Step 2. This modification of P makes the observation table contain whether $w \in L$.

2.1.2 Balle and Mohri's Weighted Automata Learning

Balle and Mohri's algorithm [12] is an extension of the L^* algorithm, and extracts a *weighted finite automaton* (instead of a deterministic finite automaton) from a *rational weighted language* (instead of a regular language).

2.1.2.1 Weighted Finite Automaton (WFA)

■Definition A *weighted finite automaton (WFA)* is an automaton to define a *weighted language*, which is a function taking a word and returning a value in a semiring. In Chapter 5, this semiring will be assumed to be the real field $\mathbb{R}$.

Definition 2.1 (weighted language)**.** *A* weighted language *is a function of type* $\Sigma^* \to \mathbb{S}$*, where* $\mathbb{S}$ *is a semiring and* Σ *is an alphabet.*

Definition 2.2 (Weighted finite automaton (WFA))**.** *A* weighted finite automaton (WFA) *over a semiring* $\mathbb{S}$ *is a quadruple* $A = (\Sigma, \alpha, \beta, (A_\sigma)_{\sigma \in \Sigma})$ *of:*

- *a finite set called* alphabet*:* Σ*,*
- *an* initial vector*:* $\alpha \in \mathbb{S}^d$*,*
- *a* final vector*:* $\beta \in \mathbb{S}^d$*,*
- *a* transition matrix*:* $(A_\sigma)_{\sigma\in\Sigma} \in \mathbb{S}^{d\times d}$ *(assuming the WFA is named A).*

The natural number d *is referred to as* the number of states*.*

A WFA A induces a weighted language $f_A \colon \Sigma^* \to \mathbb{S}$.

Definition 2.3 (induced weighted language of a WFA / rational weighted language)**.** *Let* A *be the WFA in Definition 2.2. The* weighted language *induced by* A *is*

$$f_A(w_1 \dots w_n) = \alpha^\top A_{w_1} \dots A_{w_n} \beta. \tag{2.10}$$

A weighted language $f \colon \Sigma^* \to \mathbb{S}$ *is* rational *if and only if it is induced by a WFA.*

The intuition of a run of the WFA A is to put the initial vector in the memory first. For each input character $\sigma \in \Sigma$, the vector in the memory is multiplied by the corresponding transition matrix A_σ. When the input sequence (word) reaches EOF, the inner product of the vector in the memory and the final vector becomes the result of the run.

We refer the memory state in the running process to as the *configuration*.

Definition 2.4 (configuration of a WFA)**.** *Let* A *be a WFA as in Definition 2.2. The* configuration function *of* A *is*

$$\delta_A^*(w_1 \dots w_n) = \alpha^\top A_{w_1} \dots A_{w_n}. \tag{2.11}$$

The configuration of A *at a word* $w_1 \dots w_n$ *is defined by* $\delta_A^*(w_1 \dots w_n)$*.*

Obviously, $f_A(w_1 \dots w_n) = \delta_A^*(w_1 \dots w_n)\beta$ holds.

Example 2.5. *Let* $A = (\Sigma, \alpha, \beta, (A_\sigma)_{\sigma\in\Sigma})$ *be the following WFA over the real field* $\mathbb{R}$*:*

$$\Sigma = \{\, a, b \,\}, \alpha = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \beta = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, A_a = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 1 & 1 \\ 0 & 3 & 1 \end{pmatrix}, A_b = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & -1 & -2 \end{pmatrix}. \tag{2.12}$$

The run of the WFA A *on the word "ab" is:*

$$\delta_A(\text{“}ab\text{”}) = \alpha^\top A_a A_b = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & -1 & 0 \\ 2 & 1 & 1 \\ 0 & 3 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & -1 & -2 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}, \tag{2.13}$$

$$f_A(\text{“}ab\text{”}) = \delta_A(\text{“}ab\text{”})\beta = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = 1. \tag{2.14}$$

Table 2.2: The table of $f'_A(e_i, w)$ ($i = 1, 2, \ldots, d$ and $w \in \Sigma^*$) of Example 2.5

configuration $\setminus$ w	ϵ	a	b	aa	ab	ba	bb	...
$(1, 0, 0)$	0	-1	2	-3	1	6	-2	...
$(0, 1, 0)$	1	2	1	4	2	1	3	...
$(0, 0, 1)$	1	4	-3	10	0	-11	7	...

A WFA can also be interpreted graphically. In the graphical interpretation, the number of states corresponds to the number of nodes, and the entry at the i-th row and j-th column of the transition matrix $A_\sigma (\sigma \in \Sigma)$ corresponds to the weight of the σ-labeled edge from the i-th node to the j-th node. The entries of the final vector β are written on the nodes as "weighted accepting state". At the beginning of the run, α_i number of tokens are put on the i-th node. For each input character σ and for each node, the token placed there is multiplied by the weight of the edge labeled by σ, and placed on the destination node. The tokens on the node are accumulated by the addition of the semiring $\mathbb{S}$. When the input word reaches EOF, the result is the sum of the multiplications of the number of tokens on the nodes and the corresponding entry of β. The graphical interpretation of Example 2.5 is in Figure 2.2.

■Linear Dependency of the Configurations A theory similar to the equivalence of the states of a DFA can be developed for WFAs. For a DFA $A = (\Sigma, Q, \delta, q_0, F)$, two states q and q' are *equivalent* if and only if $\delta^*(q, w) \in F \iff \delta^*(q', w) \in F$ for all words $w \in \Sigma^*$. If q and q' are not equivalent, then they are *distinguishable*. When they are equivalent, we can merge them, and shrink the number of states while preserving the behavior of A. We obtain the minimized DFA of A by repeating mergings.

To discuss the "equivalence" of configurations of WFAs, we define $f'_A \colon \mathbb{R}^d \times \Sigma^* \to \mathbb{R}$ as:

$$f'_A(v, w_1 \ldots w_n) = v A_{w_1} A_{w_2} \ldots A_{w_n}. \tag{2.15}$$

Obviously, $f_A(w) = f'_A(\alpha^\top, w)$ holds for all words $w \in \Sigma^*$. Intuitionally, f'_A resumes the run from a given configuration. We define the equivalence of the configurations v and v' as $f'_A(v, w) = f'_A(v', w)$ for all words $w \in \Sigma^*$.

We can consider the table of $f'_A(e_i, w)$ for $i = 1, 2, \ldots, d$ and $w \in \Sigma^*$ as we consisdered $\delta^*(q, w)$ in the discussion on DFAs. For example, the table for Example 2.5 is shown in Table 2.2.

A relation

$$(\text{3rd row}) = (-2)(\text{1st row}) + (\text{2nd row}) \tag{2.16}$$

is observed in this case, and it suggests that the 3rd state is redundant. In the graphical

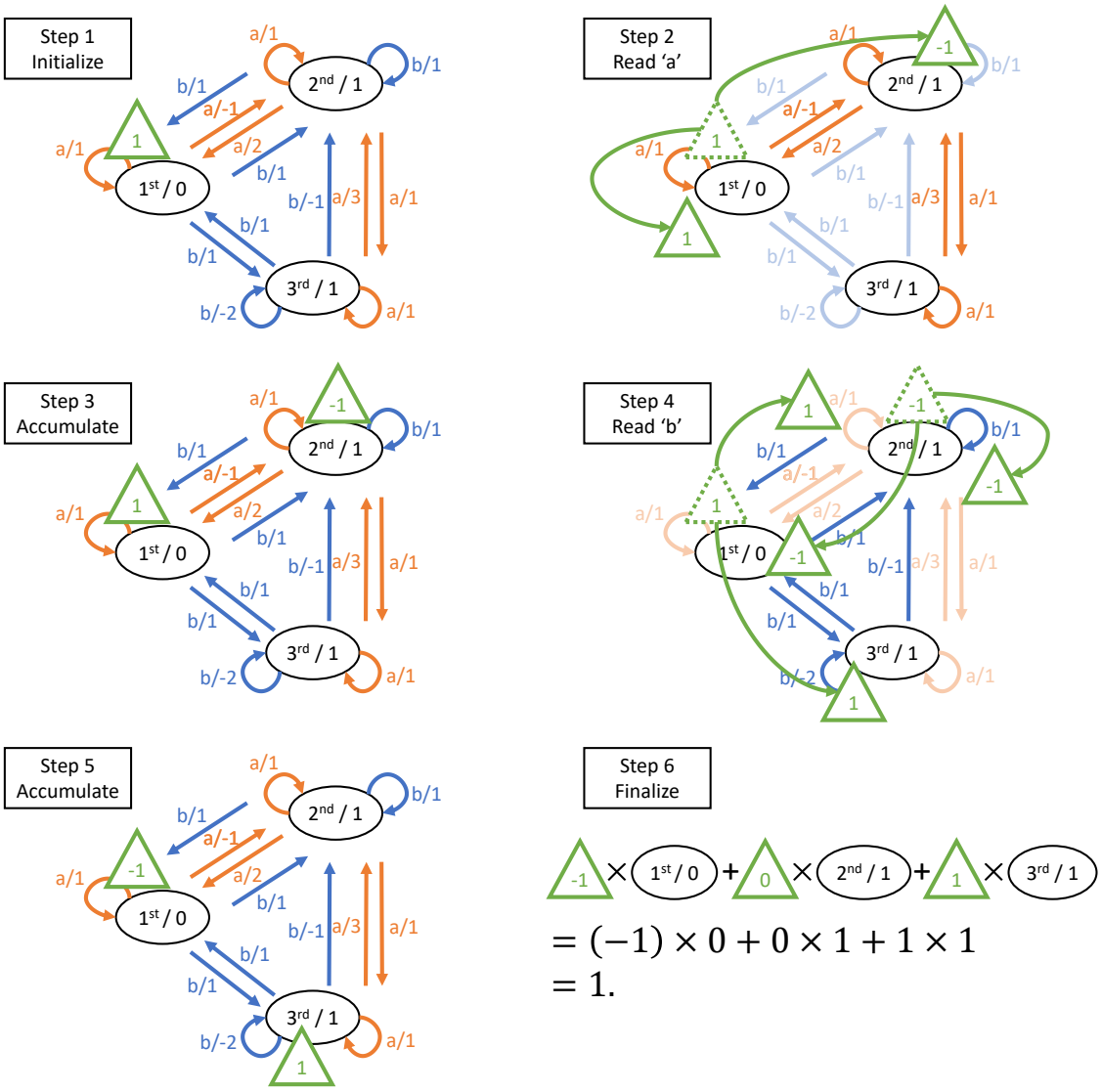


Figure 2.2: The graphical interpretation of the run of Example 2.5. The transition of "a" is drawn with red arrows, and the transition of "b" is drawn with blue arrows. The tokens on the nodes are written by green triangles. The texts on the nodes are of form "(ID of the node)/(the corresponding entry of β)".

Table 2.3: The behavior of WFA A'

ϵ	a	b	aa	ab	ba	bb	...
0	-1	2	-3	1	6	-2	...

interpretation, putting one token on the 3rd state is equivalent to putting (-2) number of tokens on the 1st state and one token on the 2nd state. By reducing the 3rd state and reconnecting the transitions, we obtain a WFA $A' = (\Sigma, \alpha', \beta', (A'_\sigma)_{\sigma\in\Sigma})$:

$$\alpha' = \begin{pmatrix}1\\0\end{pmatrix}, \beta' = \begin{pmatrix}0\\1\end{pmatrix}, A'_a = \begin{pmatrix}1 & -1\\0 & 2\end{pmatrix}, A'_b = \begin{pmatrix}-2 & 2\\1 & 1\end{pmatrix} \tag{2.17}$$

The behavior of A' is shown in Table 2.3. Comparing the first row of Table 2.2 and Table 2.3 shows that the behaviors of A and A' are the same.

We can interpret this reconnecting by matrix operations as follows. Let matrices $X \in \mathbb{R}^{2\times 3}$ and $Y \in \mathbb{R}^{3\times 2}$ be

$$X = \begin{pmatrix}1 & 0 & 0\\0 & 1 & 0\end{pmatrix}, Y = \begin{pmatrix}1 & 0\\0 & 1\\-2 & 1\end{pmatrix}. \tag{2.18}$$

The matrix X shows the translating rule of tokens from A' to A: The first row means that putting one token on the first state of A' corresponds to putting one token on the first state of A. The second row means the similar rule. As we are getting rid of the third state of A, the third column of X is filled with zero. The matrix Y shows the translating rule of tokens from A to A': The first row means that putting one token on the first state of A corresponds to putting one token on the first state of A'. The second row means the similar rule. The third row means that putting one token on the third state of A corresponds to putting (-2) tokens on the first state and putting one token on the second state of A' (remember Equation 2.16). Applying these translating rules to A as

$$(\alpha')^\top = \alpha^\top Y, \beta' = X\beta, A'_\sigma = XA_\sigma Y \quad \text{for each } \sigma \in \Sigma \tag{2.19}$$

yields the WFA A' above. For a formal discussion about reducing unnecessary states, see Section 6.4.2.

The same procedure to reduce the number of states cannot be applied to A'. Such WFAs are called *minimal*.

Definition 2.6 (minimal WFA)**.** *A WFA A with d states is* minimal *if there exists no WFA A' with $d' < d$ states on the same alphabet such that $f_A = f'_A$.*

Though the expression of the minimal WFA of an arbitrary WFA is not unique, it is unique up to change of basis. For the theory of minimality, see [13].

As the above discussion suggests, the linear independency (and dependency) of $\lambda w.f'(v,w)$ for each $v \in \mathbb{S}^d$ plays an important role in the theory of WFAs. Balle and Mohri's WFA learning algorithm works by taking advantage of this fact.

2.1.2.2 Problem Formulation

As the L*algorithm uses a teacher to learn a black box regular language, Balle and Mohri's algorithm uses a teacher to learn a black box rational weighted language over $\mathbb{R}$. Remark that the algorithm does not work for general semirings.

Formally, Balle and Mohri's algorithm takes as inputs an oracle answering the membership query $m\colon \Sigma^* \to \mathbb{R}$ and an oracle answering the equivalence query $e\colon \{\,\text{WFAs}\,\} \to \{\,\texttt{Equivalent}\,\} \sqcup \Sigma^*$ and outputs a minimal WFA A. The relationship between the two input oracles and the output minimal WFA is as follows:

> Let $f\colon \Sigma^* \to \mathbb{R}$ be a rational weighted language. If m satisfies
>
> $$m(w) = f(w), \tag{2.20}$$
>
> and e satisfies
>
> $$e(A') = \begin{cases} \texttt{Equivalent} & ; f_{A'} = f, \\ w & ; f_{A'}(w) \neq f(w) \end{cases} \tag{2.21}$$
>
> then the algorithm terminates, and $f = f_A$ holds.

The situation is almost the same as L* algorithm explained in Section 2.1.1.1, but remark that m is of form $\Sigma^\star \to \mathbb{R}$ (asking "What is the value $f(w)$ for a word w?"), not $\Sigma^\star \times \mathbb{R} \to \{\,\texttt{tt}, \texttt{ff}\,\}$ (asking "For a word w, is the value $f(w)$ equivalent to this value?").

2.1.2.3 Principle

■Constructing a WFA from a Hankel matrix As we defined the Hankel matrix for a language, we define the Hankel matrix for a weighted language. The Hankel matrix of a weighted language $f\colon \Sigma^* \to \mathbb{R}$ is a table of infinite numbers of rows and columns labeled by words in Σ^* with the p-th row and s-th column indicating the value $f(p \cdot s)$. For example, the Hankel matrix of the weighted language of Example 2.5 is shown in Table 2.4.

We saw in Section 2.1.1.3 that the row contents of the Hankel matrix of a regular language are finite. The Hankel matrix of a rational weighted language also has a "good" property about the row contents. It follows from the following theorem.

Table 2.4: Hankel matrix of the weighted language of Example 2.5

prefix\suffix	ϵ	a	b	ab	$\cdots$	combination of the rows
ϵ	0	-1	2	1	$\cdots$	(ϵ-th row)
a	-1	-3	1	-1	$\cdots$	(a-th row)
b	2	6	-2	2	$\cdots$	(-2)(a-th row)
aa	-3	-7	-1	-5	$\cdots$	(-2)(ϵ-th row) + 3(a-th row)
ab	1	5	-5	-1	$\cdots$	(-2)(ϵ-th row) + (-1)(a-th row)
aba	5	13	-1	7	$\cdots$	2(ϵ-th row) + (-5)(a-th row)
abb	-5	-17	9	-3	$\cdots$	2(ϵ-th row) + 5(a-th row)
abaa	13	29	7	23	$\cdots$	20(ϵ-th row) + (-13)(a-th row)
abab	-1	-13	21	9	$\cdots$	10(ϵ-th row) + (a-th row)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	

Theorem 2.7 (Fliess [48])**.** *Let $\mathbb{S}$ be a semiring. A weighted language $f\colon \Sigma^* \to \mathbb{S}$ is rational if and only if the linear space spanned by the rows of its Hankel matrices is finitely generated.*

Of course, if $\mathbb{S}$ is the real field $\mathbb{R}$, the latter condition means the rank of the Hankel matrix is finite. We can observe that the rank of the Hankel matrix shown in Table 2.4 is two, and it reflects the number of states of the WFA of Example 2.5.

Based on this theory, we can extract the WFA from the Hankel matrix in the following procedure:

1. Pick row labels $w_1, \ldots, w_n \in \Sigma^*$ from the Hankel matrix so that the corresponding rows form a basis of the linear space generated by the rows. Let the number of states of the target WFA be n. In the graphical representation, we make n number of states and label the states by the row labels.
2. For each $i = 1, \ldots, n$ and $\sigma \in \Sigma$, the i-th row $(c_1, \ldots, c_n)$ of the transition matrix A_σ is determined by the relation on the rows of the Hankel matrices

$$(\text{the } w_i\sigma\text{-th row}) = c_1(\text{the } w_1\text{-th row}) + \cdots + c_n(\text{the } w_n\text{-th row}). \tag{2.22}$$

   In the graphical representation, this step calculates the weights of the transitions departing i-th node with label σ.
3. The initial vector $\alpha = (\alpha_1, \ldots, \alpha_n)^\top \in \mathbb{R}^n$ is determined by the relation on the rows of the Hankel matrices

$$(\text{the } \epsilon\text{-th row}) = \alpha_1(\text{the } w_1\text{-th row}) + \cdots + \alpha_n(\text{the } w_n\text{-th row}). \tag{2.23}$$

4. The i-th element of the final vector $\beta \in \mathbb{R}^n$ is the entry at the w_i-th row and ϵ-th column of the Hankel matrix.

For example, we can get a WFA $A'' = (\Sigma, \alpha'', \beta'', (A''_\sigma)_{\sigma\in\Sigma})$ by the procedure in Example 2.5 as follows: We pick row labels ϵ and a (of course, the choice is arbitrary). The transition matrices can be determined by looking at "combination of the rows" in Table 2.2 as

$$A''_a = \begin{pmatrix} 0 & 1 \\ -2 & 3 \end{pmatrix}, A''_b = \begin{pmatrix} 0 & -2 \\ -2 & -1 \end{pmatrix}. \tag{2.24}$$

The initial vector α'' is $(1, 0)$ since ϵ itself is picked in the first step. The final vector β'' is $(0, -1)$ as the ϵ-th and "a"-th rows show. Though the represenation is different from Example 2.5, the induced weighted language is the same. The equivalence of A' and A'' is easily checked by the relations

$$(\alpha')^\top = (\alpha'')^\top T, \quad \beta' = T^{-1}\beta'', \quad A'_a = T^{-1}A''_a T, \quad A'_b = T^{-1}A''_b T, \tag{2.25}$$

where

$$T = T^{-1} = \begin{pmatrix} 1 & 0 \\ 1 & -1 \end{pmatrix}. \tag{2.26}$$

■Constructing a WFA from a Hankel submatrix (observation table) As we did in Section 2.1.1.3, we define Hankel submatrices, which are finite versions of the Hankel matrix, for WFAs and consinder the condition, which is called *completeness*, that the procedure above works. A *Hankel submatrix* of a weighted language $f\colon \Sigma^* \to \mathbb{R}$ is a table of row labels P and column labels S such that its p-th row and s-th column indicates the value $f(p \cdot s)$, where P is a finite prefix-closed set and S is a finite suffix-closed set.

When a Hankel submatrix is *incomplete*, we cannot construct a WFA from the Hankel submatrix because of the lack of data.

Definition 2.8 (complete/incomplete)**.** *A Hankel submatrix with row labels P and column labels S is* complete *if and only if the rank of the Hankel submatrix is equal to the rank of a Hankel matrix whose row labels are extended to $P \cup P\Sigma$. If it is not complete, it is* incomplete*.*

Intuitionally, it means for any configuration of the WFA and any character $\sigma \in \Sigma$, the configuration after reading the character σ can be represented by a superposition of the knwon configurations. This *completeness* property corresponds to the closedness in the DFA case.

To resolve the incompleteness, find $p \in P$ and $\sigma \in \Sigma$ such that `rank`(the Hankel submatrix) $\neq$ `rank`(the Hankel matrix whose row label set is extended to $P \cup \{\, p\sigma \,\}$), and add $p\sigma$ to P.

■Algorithm The idea is the same as Section 2.1.1.3: start with the Hankel submatrix with $P = \{\epsilon\}, S = \{\epsilon\}$, and extend it so that it becomes complete. The procedure overall is:

1. Let the row label set $P = \{\epsilon\}$ and the column label set $S = \{\epsilon\}$, and fill the Hankel submatrix by raising membership queries.
2. Check the completeness of the Hankel submatrix determined by (P, S). If it is not complete, extend the row label set P by the procedure above. Repeat this step until the Hankel submatrix becomes complete.
3. Construct a WFA from the complete Hankel submatrix.
4. Raise an equivalence query by the constructed WFA.
5. If the answer is `Equivalent`, the learning is done. If the answer is a counterexample $w \in \Sigma^*$, find the longest prefix $p \in P$ of w, decompose w into $p\sigma s$, where $\sigma \in \Sigma$ and $s \in \Sigma^*$, and add all the suffixes of s into S. This modification of S makes the Hankel submatrix contain the value for the counterexample w. Go back to Step 2.

2.2 IMPACT Algorithm: Program Verification Using Interpolants

As the preliminary for Chapter 3 and Chapter 4, whose interests are in interpolants, we introduce the IMPACT algorithm [89], which uses interpolation for program verification. Note that we use our own notations to make this section self-contained. We also fixed some seeming errors in the original paper.

2.2.1 Overview of the IMPACT Algorithm

The IMPACT algorithm takes a *control flow graph (CFG)* and verifies whether it reaches the unsafe state with interpolant computation. A CFG is a graph expressing a program with assignment and branching edges. Specifically, a CFG is an LTS that consists of nodes and labeled edges, both with unique IDs. The internal states of a CFG are the position of the token and the assignments of the variables. There are two types of edges: edges for variable assignments and edges for condition branching. When an edge for variable assignments is chosen, the variables are updated by the assignment command on the edge. When an edge for condition branching is chosen, and the current variables do not satisfy the condition on the edge, the variables are updated to be $\bot$. This behavior means an edge can be chosen only when the current variables satisfy the condition. Hence the safety of a CFG is that the variable assignments are $\bot$ when the token reaches the unsafe nodes on the CFG. Figure 2.3 shows an example of the CFG given in [89].

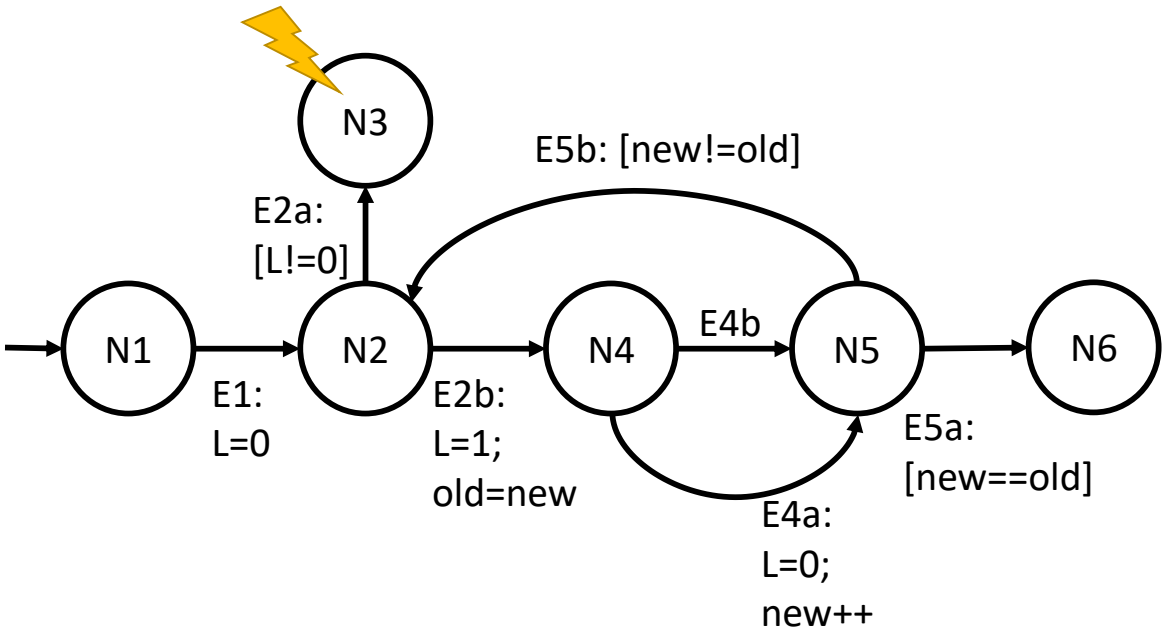


Figure 2.3: A control flow graph given in [89]. Unsafe nodes are marked by lightning. The IDs of nodes are written as `N**`. The IDs of edges are written as `E**`. The assignments are written along the edges. The conditions of condition branches are written along the edges with brackets. Note that edge `E4b` do nothing.

The IMPACT algorithm aims to construct a graph called an *unwinding graph* as a proof of safety. An unwinding graph is informally an expansion of the executions of the CFG into a tree. It is equipped with *covering edges* to go back to an ancestor from a descendant, and each node has an *overapproximation* of the state of the variables. An unwinding graph is formally described as the following conditions C1-C10:

C1 The number of nodes and edges is finite.

C2 A node has a *CFG-ID*, which is the ID of a node of the CFG. Multiple nodes with the same CFG-ID can exist.

C3 Each node is equipped with a predicate over the variables. This predicate is called an *overapproximation*.

C4 Each edge is either a *transition edge* or a *covering edge*.

C5 Each transition edge is equipped with the ID of a CFG edge. Multiple transition edges with the same ID can exist.

C6 The subgraph formed by the transition edges is a tree whose root has the ID of the initial node of the CFG.

C7 The path made by following transition edges starting from the root is an execution path in the CFG.

C8 The ID on the source of a covering edge is the same as the ID on the target of the edge.

C9 The predicate on the source of a covering edge implies the predicate on the target of the edge.

C10 For any covering edge e and for any descendant (including itself) v of the source of e, there exists no covering edge e' such that the target of e' is v.

We can interpret an unwinding graph as an LTS as follows: It takes the IDs of the edges of the CFG as inputs, and outputs a proposition about the variables. At the beginning of the execution of the unwinding graph, the token is put on the root. The token moves along the edge corresponding to the input. When the token reaches the source of a covering edge, the token is immediately moved to the target of the edge. It returns the property on the node as the output.

In this interpretation, the token never reach the descendants (including itself) of the sources of the covering edges, and the descendants are "not necessary." The descendants (including itself) of the covering edges are called *covered*. The condition C10 prevents the token from moving to a covered node by a covering edge.

We are defining "expandedness" to describe the condition that an unwinding graph can mimic the behavior of a CFG. A node v of an unwinding graph is called *expanded* if

$$\{\,\text{ID of } e \mid e \text{ is a transition edge whose source is } v\,\} \tag{2.27}$$
$$= \{\,\text{ID of } e \mid e \text{ is an edge of the CFG whose source is the edge corresponding to } v\,\} \tag{2.28}$$

holds. If the token is on an expanded node, the unwinding graph can mimic any transition of the CFG.

The condition of an unwinding graph to mimic any transitions of a CFG is described as follows:

C11 Any node of the unwinding graph is covered or expanded.

The IMPACT algorithm constructs an unwinding graph from a CFG satisfying the following property:

C12 For any execution path of the CFG, the state of the variables determined by the execution path satisfies the output of the unwinding graph.

If the constructed unwinding graph from a CFG satisfies C1-C12 and the following C13, it works as a proof of safety of the CFG.

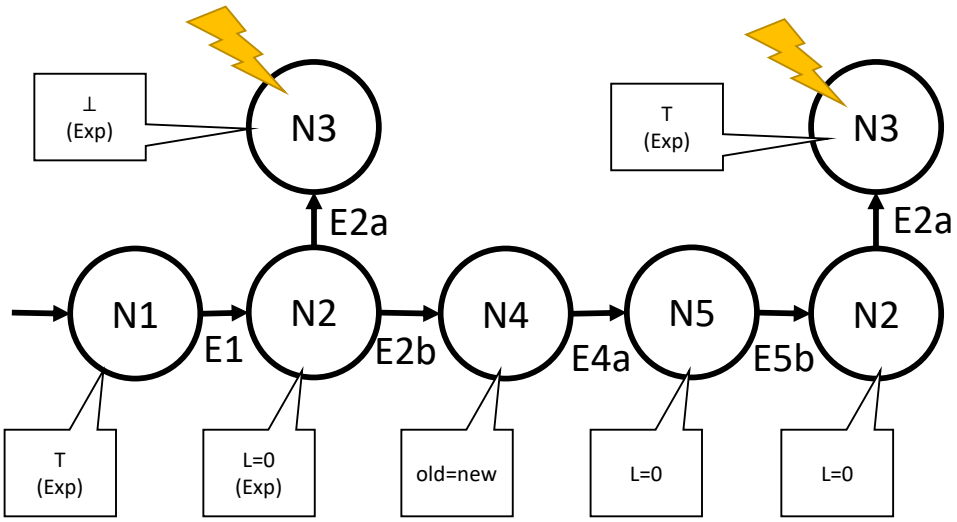


Figure 2.4: An unwinding graph satisfying C1-10 and C12-C13 of the CFG in Figure 2.3. Unsafe nodes are marked by lightning. Conditions on nodes and whether nodes are expanded are written with balloons.

C13 The predicate on any non-covered node corresponding to an unsafe node is $\bot$.

Figure 2.4 shows an example of an unwinding graph satisfying C1-10 and C12-C13 of the CFG in Figure 2.3. Note that the unwinding graph does not satisfy C11 because some nodes are non-expanded and non-covered. Figure 2.5 shows an example of an unwinding graph satisfying C1-13.

The IMPACT algorithm tries to construct an unwinding graph from a given CFG by a depth-first search. If an unwinding graph satisfying C1-13 is found during the search, it returns "safe." If the reach to the unsafe node is found during the search, it returns "unsafe" with a witness.

2.2.2 Flow

The flow of the IMPACT algorithm is shown in Figure 2.6.

Step 1 Make an initial unwinding graph that consists of one node. The predicate associated with the node is $\top$, which obviously satisfies the conditions C1-C10 and C12. The ID of the node is the ID of the initial node of the target CFG.

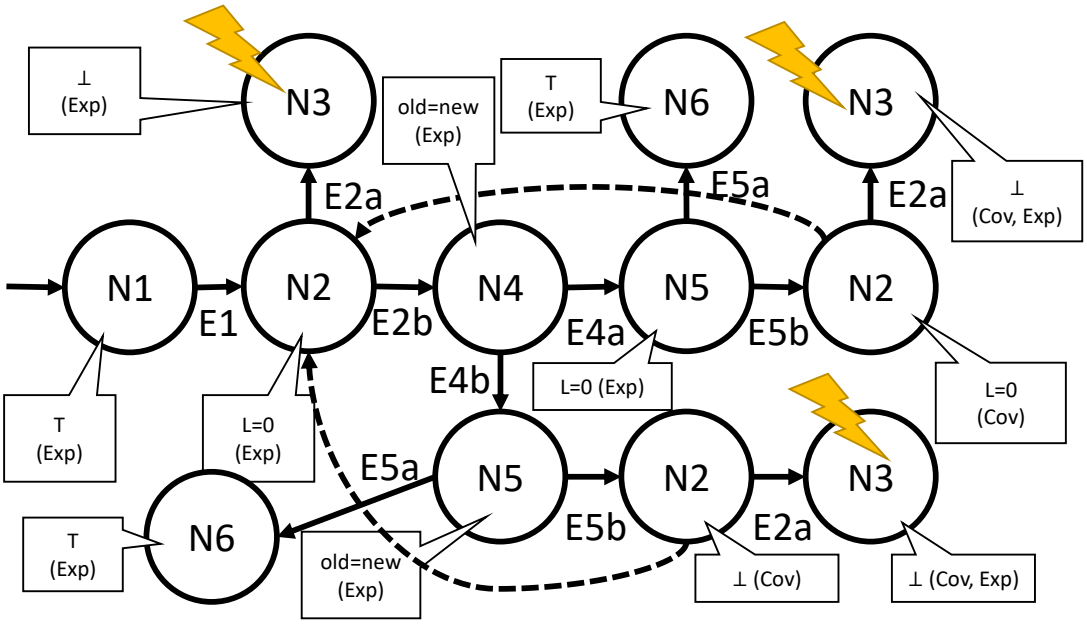


Figure 2.5: An unwinding graph satisfying C1-13 of the CFG in Figure 2.3. Unsafe nodes are marked by lightning. Covering edges are written with dotted arrows. Conditions on nodes and whether nodes are expanded and covered are written with balloons.

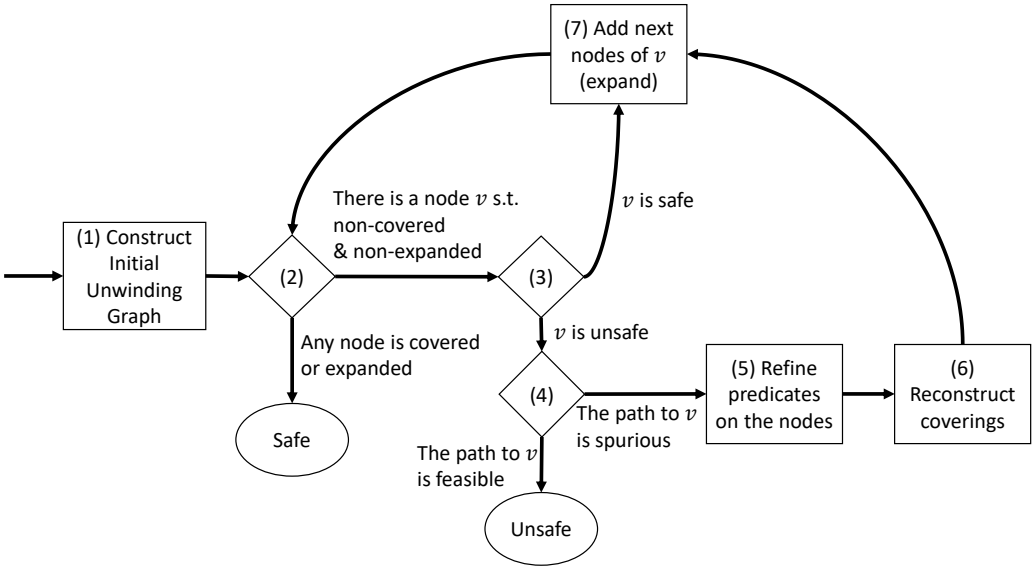


Figure 2.6: The flow of IMPACT algorithm

Step 2 Pick a non-covered and non-expanded node v from the unwinding graph (as a step of the DFS). If there is no such node, it means the unwinding graph is constructed to satisfy the conditions C1-C13, and the safety is proved.

Step 3 If the CFG node whose ID is the CFG-ID of v is unsafe, go to Step 4 to investigate if the corresponding execution path is feasible. If it is not, go to Step 7 to continue the search.

Step 4 Check if the execution path corresponding to the path on the unwinding graph from the root to the node v actually happens or not. If it does not happen, the path is called spurious. Spurious paths can exist because the construction of the unwinding graph is ignoring the branch conditions of the target CFG. This step is explained in Section 2.2.3.

Step 5 Strengthen the predicates on the nodes so that the condition C12 is satisfied and the predicate on the unsafe node is $\bot$.

Step 6 Construct covering edges so that the conditions C8, C9, C10 are satisfied.

Step 7 Look at the node v' of the CFG corresponding to v, get the edges whose sources are v', and copy the edges into the unwinding graph so that the sources are v and the targets are fresh nodes. The predicates associated with the fresh nodes are $\top$. After this step, the node v is expanded.

2.2.3 Refinement Using Interpolants

We are explaining how we use SMT solvers and interpolant computation in Steps 5 and 6 in the previous section. For the node v fixed in Step 3, we get the corresponding execution path from the path on the unwinding graph from the root to v. To determine whether the execution path can happen, we convert the operations of the execution path into a sequence of predicates. We introduce indexed variables $x_0, x_1, \ldots$ for a variable x, and make them represent the changes of the value of x. Specifically, the assignment operation $y = f(y, x^{(1)}, x^{(2)}, \ldots, x^{(n)})$ is converted into the predicate $y_{i+1} = f(y_i, x_{j_1}^{(1)}, \ldots, x_{j_n}^{(n)})$, where i is the index of y and j_k is the index of $x^{(k)}$, and the branching $f(x^{(1)}, \ldots, x^{(n)}) = g(x^{(1)}, \ldots, x^{(n)})$ is converted into the predicate $f(x_{i_1}^{(1)}, \ldots, x_{i_n}^{(n)}) = g(x_{i_1}^{(1)}, \ldots, x_{i_n}^{(n)})$.

Assume that we obtain the sequence of predicates $P_1, \ldots, P_n$ by the conversion above for the path on the unwinding graph

$$v_0 \xrightarrow{o_1} v_1 \xrightarrow{o_2} \ldots \xrightarrow{o_n} v_n. \tag{2.29}$$

For each i, let Q_i be a predicate satisfying $P_1 \wedge \cdots \wedge P_i \Rightarrow Q_i$, and R_i be the predicate obtained by removing indices from Q_i. The state of the variables after executing $o_1, \ldots, o_n$

Table 2.5: The table of i, Q_i, and R_i

i	A	B	Q_i: Interpolant of A and B	R_i: Generated predicate
0	(Empty)	$P_1 \wedge \cdots \wedge P_5$	$\top$	$\top$
1	P_1	$P_2 \wedge \cdots \wedge P_5$	$\top$	$\top$
2	$P_1 \wedge P_2$	$P_3 \wedge P_4 \wedge P_5$	$\mathtt{old}_1 = \mathtt{new}_0$	$\mathtt{old} = \mathtt{new}$
3	$P_1 \wedge P_2 \wedge P_3$	$P_4 \wedge P_5$	$\mathtt{L}_2 = 0$	$\mathtt{L} = 0$
4	$P_1 \wedge \ldots P_4$	P_5	$\mathtt{L}_2 = 0$	$\mathtt{L} = 0$
5	$P_1 \wedge \ldots P_5$	(Empty)	$\bot$	$\bot$

on the CFG satisfies R_i by construction. Hence we can refine the predicates on the nodes by strengthening with R_i so that the unwinding graph satisfies the condition C12. [89] uses interpolant computation to find such predicates, as the interpolant of $P_1 \wedge \cdots \wedge P_i$ and $P_{i+1} \wedge \cdots \wedge P_n$ satisfies the condition of Q_i.

For example, assume that the unsafe path of the CFG in Figure 2.3

$$N1 \xrightarrow{E1} N2 \xrightarrow{E2b} N4 \xrightarrow{E4a} N5 \xrightarrow{E5b} N2 \xrightarrow{E2a} N3 \tag{2.30}$$

is picked in Step 4. The operations are converted into the list of predicates

$$\underbrace{\mathtt{L}_0 = 0}_{P_1} \to \underbrace{\mathtt{L}_1 = 1 \wedge \mathtt{old}_1 = \mathtt{new}_0}_{P_2} \to \underbrace{\mathtt{L}_2 = 0 \wedge \mathtt{new}_1 = \mathtt{new}_0 + 1}_{P_3} \to \underbrace{\mathtt{new}_1 \neq \mathtt{old}_1}_{P_4} \to \underbrace{\mathtt{L}_2 \neq 0}_{P_5}. \tag{2.31}$$

We obtain Table 2.5 by applying the procedure above to the list. By using these predicates $R_0, \ldots, R_5$, we strengthen the predicates on the nodes of the unwinding graph.

Chapter 3

Sharper and Simpler Nonlinear Interpolants for Program Verification

This chapter is based on joint work [102] with Yuki Nishida, Kensuke Kojima, Kohei Suenaga, Kengo Kido, and Ichiro Hasuo.

3.1 Introduction

3.1.1 Interpolation for Program Verification

Interpolation in logic is a classic problem. Given formulae φ and ψ that are jointly unsatisfiable (meaning $\models \varphi \wedge \psi \Rightarrow \bot$), one asks for a "simple" formula ξ such that $\models \varphi \Rightarrow \xi$ and $\models \xi \wedge \psi \Rightarrow \bot$. The simplicity requirement on ξ can be a formal one (like the *common variable condition*, see Definition 3.10) but it can also be informal, like "ξ is desirably much simpler than φ and ψ (that are gigantic)." Anyway, the intention is that ξ should be a simple witness for the joint unsatisfiability of φ and ψ, that is, an "essential reason" why φ and ψ cannot coexist.

This classic problem of interpolation has found various applications in static analysis and program verification [55, 61, 69, 89–91]. This is particularly the case with techniques based on automated reasoning, where one relies on symbolic *predicate abstraction* in order to deal with infinite-state systems like (behaviors of) programs. It is crucial for the success of such techniques that we discover "good" predicates that capture the essence of the systems' properties. Interpolants—as simple witnesses of incompatibility—have proved to be potent candidates for these "good" predicates.

3.1.2 Interpolation via Optimization and Real Algebraic Geometry

A lot of research efforts have been made towards efficient interpolation algorithms. One of the earliest is [34]: it relies on *Farkas' lemma* for synthesizing linear interpolants (i.e., interpolants expressed by linear inequalities). This work and subsequent ones have signified the roles of *optimization* problems and their algorithms in efficient synthesis of interpolants.

In this line of research we find the recent contribution by Dai et al. [39] remarkable, from both theoretical and implementation viewpoints. Towards synthesis of *nonlinear* interpolants (that are expressed by polynomial inequalities), their framework in [39] works as follows.

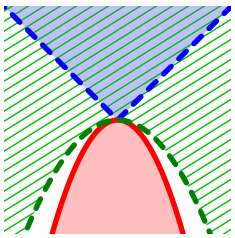
- On the theory side it relies on *Stengle's Positivstellensatz*—a fundamental result in *real algebraic geometry* [18, 124]—and relaxes the interpolation problem to the problem of finding a suitable "disjointness certificate." This certificate consists of a few polynomials subject to certain conditions.
- On the implementation side it relies on state-of-the-art *SDP solvers* to efficiently solve the SDP problem that results from the above relaxation.

In [39] it is reported that the above framework successfully synthesizes nontrivial nonlinear interpolants, where some examples are taken from program verification scenarios.

3.1.3 Contribution

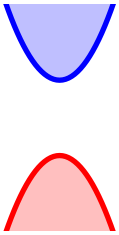
The current work contributes an enhancement of the framework from [39]. Our specific concerns are in *sharpness* and *simplicity* of interpolants.

Example 3.1 (sharp interpolant) Let $\mathcal{T} \coloneqq (y > x \wedge x > -y)$ and $\mathcal{T}' \coloneqq (y \leq -x^2)$. These designate the blue and red areas in the figure, respectively. We would like an interpolant $\mathcal{S}$ so that $\mathcal{T}$ implies $\mathcal{S}$ and $\mathcal{S}$ is disjoint from $\mathcal{T}'$. Note however that such an interpolant $\mathcal{S}$ must be "sharp." The areas of $\mathcal{T}$ and $\mathcal{T}'$ almost intersect with each other at $(x, y) = (0, 0)$. That is, the conditions $\mathcal{T}$ and $\mathcal{T}'$ are barely disjoint in the sense that, once we replace $>$ with $\geq$ in $\mathcal{T}$, they are no longer disjoint. (See Definition 3.16 for formal definitions.)



The original framework in [39] fails to synthesize such "sharp" interpolants; and this failure is theoretically guaranteed (see Section 3.3.1). In contrast our modification of the framework

succeeds: it yields an interpolant $8y + 4x^2 > 0$ (the green hatched area).



Example 3.2 (simple interpolant) Let $\mathcal{T} \coloneqq (y \geq x^2 + 1)$ and $\mathcal{T}' \coloneqq (y \leq -x^2 - 1)$. The implementation `aiSat` [37] of the workflow in [39] succeeds and synthesizes an interpolant $284.3340y + 0.0012x^2y > 0$. In contrast our tool synthesizes $5y + 2 > 0$ that is much simpler.

The last two examples demonstrate two issues that we found in the original framework in [39]. Our enhanced framework shall address these issues of sharpness and simplicity, employing the following two main technical pieces.

The first piece is *sharpened Positivstellensatz-inspired relaxation* (Section 3.3). We start with the relaxation in [39] that reduces interpolation to finding polynomial certificates. We devise its "sharp" variant that features: the use of *strict inequalities* $>$ (instead of *disequalities* $\neq$); and a corresponding adaptation of Positivstellensatz that uses a notion we call *strict cone*. Our sharpened relaxation allows encoding to SDP problems, much like in [39].

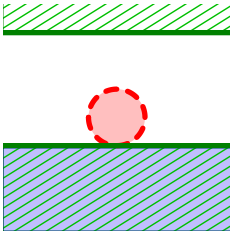
The second technical piece that we rely on is our continued fraction-based *rounding algorithm*. We employ the algorithm in what we call the *rounding-validation loop* (see Section 3.4), a workflow from [58] that addresses the challenge of *numerical errors*.

Numerical relaxation of problems in automated reasoning—such as the SDP relaxation in [39] and in the current work—is nowadays common, because of potential performance improvement brought by numerical solvers. However a numerical solution is subject to numerical errors, and due to those errors, the solution may not satisfy the original constraint. This challenge is identified by many authors [16, 58, 72, 109, 110, 114].

Moreover, even if a numerical solution satisfies the original constraint, the solution often involves floating-point numbers and thus is not simple. See Example 3.2, where one may wonder if the coefficient 0.0012 should be simply 0. Such complication is a disadvantage in applications in program verification, where we use interpolants as candidates for "useful" predicates. These predicates should grasp the essence and reflect insights of programmers; it is our hypothesis that such predicates should be simple. Similar arguments have been made in previous works such as [70, 126].

To cope with the last challenges of potential unsoundness and lack of simplicity, we employ a workflow that we call the *rounding-validation loop*. The workflow has been used e.g. in [58]; see Figure 3.1 for a schematic overview. In the "rounding" phase we apply our continued fraction-based rounding algorithm to a candidate obtained as a numerical solution of an SDP solver. In the "validation" phase the rounded candidate is fed back to the original constraints and their satisfaction is checked by purely symbolic means. If validation fails, we increment

the *depth* of rounding—so that the candidate becomes less simple but closer to the original candidate—and we run the loop again.



Example 3.3 (invalid interpolant candicate) Let $\mathcal{T} = (y \leq -1), \mathcal{T}' = (x^2+y^2 < 1)$, as shown in the figure. These are barely disjoint and hence the algorithm in [39] does not apply to it. In our workflow, the first interpolant candidate that an SDP solver yields is $f(x, y) \geq 0$, where

$$f(x,y) = \begin{pmatrix} -3.370437975 + 8.1145 \times 10^{-14}x - 2.2469y + 1.1235y^2 - 2.2607 \times \\ 10^{-10}y^3 + 9.5379 \times 10^{-11}x^2 - 2.2607 \times 10^{-10}x^2y - 4.8497 \times 10^{-11}x^2y^2 - \\ 1.1519 \times 10^{-14}x^3 + 4.8935 \times 10^{-11}x^4 - 9.7433 \times 10^{-11}y^4 \end{pmatrix} .$$

Being the output of a numerical solver the coefficients are far from simple integers. Here coefficients in very different scales coexist—for example one may wonder if the coefficient 8.1145×10^{-14} for x could just have been 0. Worse, the above candidate is in fact not an interpolant: $x = 0, y = -1$ is in the region of $\mathcal{T}$ but we have $f(0, -1) < 0$.

By subsequently applying our rounding-validation loop, we eventually obtain a candidate $34y^2 - 68y - 102 \geq 0$, and its validity is guaranteed by our tool.

This workflow of the rounding-validation loop is adopted from [58]. Our technical contribution lies in the rounding algorithm that we use therein. It can be seen as an extension of the well-known rounding procedure by *continued fraction expansion*. The original procedure, employed for example in [109], rounds a real number into a rational number (i.e., a ratio $k_1 : k_2$ between two integers). In contrast, our current extension rounds a ratio $r_1 : \cdots : r_n$ between n real numbers into a simpler ratio $k_1 : \cdots : k_n$.

We have implemented our enhancement of [39], calling our tool SSInt, (*Sharp and Simple Interpolants*) [*1]. Our experiment results support its effectiveness: the tool succeeds in synthesizing sharp interpolants (while the workflow in [39] is guaranteed to fail); and our program verification examples demonstrate the benefit of sharp and simple interpolants (synthesized by our tool) in verification. The latter benefit is demonstrated by the following example, discussed in further details later in Section 3.5.

Example 3.4 (program verification) Consider the imperative program in Listing 3.1 (pp. 65). Let us verify its assertion (the last line) by *counterexample-guided abstraction refinement (CEGAR)* [32], in which we try to synthesize suitable predicates that separate the reachable region (that is under-approximated by finitely many samples of execution traces) and the unsafe region $((xa) + 2(ya) < 0)$. The use of interpolants as candidates for such separating predicates has been advocated by many authors, including [61].

[*1] The tool is available online: `https://github.com/ERATOMMSD/polysat_aplas2017`

Let us say that the first execution trace we sampled is the one in which the while loop is not executed at all ($1 \to 2 \to 3 \to 4 \to 16$ in line numbers). Following the workflow of CEGAR by interpolation, we are now required to compute an interpolant of $\mathcal{T} := (xa = 0 \wedge ya = 0)$ and $\mathcal{T}' := ((xa) + 2(ya) < 0)$. Because $\mathcal{T}$ and $\mathcal{T}'$ are "barely disjoint" (in the sense of Example 3.1, that is, the closures of $\mathcal{T}$ and $\mathcal{T}'$ are no longer disjoint), the procedure in [39] cannot generate any interpolant. In contrast, our implementation—based on our refined use of Stengle's positivstellensatz, see Section 3.3—successfully discovers an interpolant $(xa) + 2(ya) \geq 0$. This interpolant happens to be an invariant of the program and proves its safety.

3.1.4 Related Work

Aside from the work by Dai et al. [39] on which we are based, there are several approaches to polynomial interpolation in the literature. Gan et al. [49] consider interpolation for polynomial inequalities that involve uninterpreted functions, with the restriction that the degree of polynomials is quadratic. An earlier work with a similar aim is [80] by Kupferschmid et al. Gao and Zufferey [50] study nonlinear interpolant synthesis over real numbers. Their method can handle transcendental functions as well as polynomials. Interpolants are generated from refutation, and represented as union of rectangular regions. Because of this representation, although their method enjoys δ-completeness (a notion of approximate completeness), it cannot synthesize sharp interpolants like Example 3.1. Their interpolants tend to be fairly complicated formulae, too, and therefore would not necessarily be suitable for applications like program verification (where we seek simple predicates; see Section 3.5).

Putinar's positivstellensatz [112] is a well-known variation of Stengle's positivstellensatz; it is known to allow simpler SDP relaxation than Stengle's. However it does not suit the purpose of the current chapter because: 1) it does not allow mixture of strict and non-strict inequalities; and 2) it requires a compactness condition. There is a common trick to force strict inequalities in a framework that only allows non-strict inequalities, namely to add a small perturbation. We find that this trick does not work in our program verification examples; see Section 3.5.

The problem with numerical errors in SDP solving has been discussed in the literature. Harrison [58] is one of the first to tackle the problem: the work introduces the workflow of the rounding-validation loop; the rounding algorithm used there increments a denominator at each step and thus is simpler than our continued fraction-based one. The same rounding algorithm is used in [16], as we observe in the code. Peyrl & Parrilo [109], towards the goal of sum-of-square decomposition in rational coefficients, employs a rounding algorithm by continued fractions. The difference from our current algorithm is that they apply continued fraction ex-

pansion to each of the coefficients, while our generalized algorithm simplifies the ratio between the coefficients altogether. The main technical novelty of [109] lies in the identification of a condition for validity of a rounded candidate. This framework is further extended in Kaltofen et al. [72] for a different optimization problem, combined with Gauss–Newton iteration.

More recently, an approach using a simultaneous Diophantine approximation algorithm—that computes the best approximation within a given bound of denominators—is considered by Lin et al. [85]. They focus on finding a fine rational approximation to the output of SDP solvers, and do not aim at simpler certificates. Roux et al. [114] proposes methods that guarantee existence of a solution relying on numerical solutions of SDP solvers. They mainly focus on strictly feasible problems, and therefore some of our examples in Section 3.5 are out of their scope. Dai et al. [38] address the same problem of numerical errors in the context of barrier-certificate synthesis. They use *quantifier elimination* (QE) for validation, while our validation method relies on a well-known characterization of positive semidefiniteness (whose check is less expensive than QE; see Section 3.4.2).

3.1.5 Organization of the Chapter

In Section 3.2 we review the framework in [39]. Its lack of sharpness is established in Section 3.3.1; this motivates our sharpened Positivstellensatz-inspired relaxation of interpolation in Section 3.3.3. In Section 3.4 we describe our whole workflow and its implementation, describing the rounding-validation loop and the continued fraction-based algorithm used therein. In Section 3.5 we present experimental results and discuss the benefits in program verification. In Section 3.6 we conclude this chapter. In Section 3.7 we discuss the future work. Some details are deferred to appendices.

3.2 Preliminaries

Here we review the previous interpolation algorithm by Dai et al. [39]. It is preceded by its mathematical bedrock, namely Stengle's Positivstellensatz [124].

3.2.1 Real Algebraic Geometry and Stengle's Positivstellensatz

We write $\vec{X}$ for a sequence $X_1, X_2, \ldots, X_k$ of variables, and $\mathbb{R}[\vec{X}]$ for the set of polynomials in $X_1, \ldots, X_k$ over $\mathbb{R}$. We sometimes write $f(\vec{X})$ for a polynomial $f \in \mathbb{R}[\vec{X}]$ in order to signify that the variables in f are restricted to those in $\vec{X}$.

Definition 3.5 ($\mathrm{SAS}_{\neq}$) A *semialgebraic system with disequalities ($SAS_{\neq}$)* $\mathcal{T}$, in variables

$X_1, X_2, \ldots, X_k$, is a sequence

$$\mathcal{T} \;=\; \left(\begin{array}{l} f_1(\vec{X}) \geq 0 \;,\; \ldots,\; f_s(\vec{X}) \geq 0 \;, \\ h_1(\vec{X}) = 0 \;,\; \ldots,\; h_u(\vec{X}) = 0 \end{array} \qquad g_1(\vec{X}) \neq 0 \;,\; \ldots,\; g_t(\vec{X}) \neq 0 \;, \right) \tag{3.1}$$

of *inequalities* $f_i(\vec{X}) \geq 0$, *disequalities* $g_j(\vec{X}) \neq 0$ and *equalities* $h_k(\vec{X}) = 0$. Here $f_{i'}, g_{j'}, h_{k'} \in \mathbb{R}[\vec{X}]$ are polynomials, for $i' \in [1, s], j' \in [1, t]$ and $k' \in [1, u]$.

For the SAS$_{\neq}$ $\mathcal{T}$ in (3.1) in k variables, we say $\vec{x} \in \mathbb{R}^k$ *satisfies* $\mathcal{T}$ if $f_i(\vec{x}) \geq 0$, $g_j(\vec{x}) \neq 0$ and $h_k(\vec{x}) = 0$ hold for all i, j, k. We let $[\![\mathcal{T}]\!] \subseteq \mathbb{R}^k$ denote the set of all such $\vec{x}$, that is, $[\![\mathcal{T}]\!] \coloneqq \{\, \vec{x} \in \mathbb{R}^k \mid \vec{x} \text{ satisfies } \mathcal{T} \,\}$.

Definition 3.6 (cone, multiplicative monoid, ideal) A set $C \subseteq \mathbb{R}[\vec{X}]$ is a *cone* if it satisfies the following closure properties: 1) $f, g \in C$ implies $f + g \in C$; 2) $f, g \in C$ implies $fg \in C$; and 3) $f^2 \in C$ for any $f \in \mathbb{R}[\vec{X}]$.

A set $M \subseteq \mathbb{R}[\vec{X}]$ is a *multiplicative monoid* if it satisfies the following: 1) $1 \in M$; and 2) $f, g \in M$ implies $fg \in M$.

A set $I \subseteq \mathbb{R}[\vec{X}]$ is an *ideal* if it satisfies: 1) $0 \in I$; 2) $f, g \in I$ implies $f + g \in I$; and 3) $fg \in I$ for any $f \in \mathbb{R}[\vec{X}]$ and $g \in I$.

For a subset A of $\mathbb{R}[\vec{X}]$, we write: $\mathcal{C}(A)$, $\mathcal{M}(A)$, and $\mathcal{I}(A)$ for the smallest cone, multiplicative monoid, and ideal, respectively, that includes A.

The last notions encapsulate closure properties of inequality/disequality/equality predicates, respectively, in the following sense. The definition of $[\![\mathcal{T}]\!] \subseteq \mathbb{R}^k$ is in Definition 3.5.

Lemma 3.7. *Let $\vec{x} \in \mathbb{R}^k$ and $f_i, g_j, h_k \in \mathbb{R}[\vec{X}]$.*

1. *If $\vec{x} \in [\![f_1 \geq 0, \ldots, f_s \geq 0]\!]$, then $f(\vec{x}) \geq 0$ for all $f \in \mathcal{C}(f_1, \ldots, f_s)$.*
2. *If $\vec{x} \in [\![g_1 \neq 0, \ldots, g_t \neq 0]\!]$, then $g(\vec{x}) \neq 0$ for all $g \in \mathcal{M}(g_1, \ldots, g_t)$.*
3. *If $\vec{x} \in [\![h_1 = 0, \ldots, h_u = 0]\!]$, then $h(\vec{x}) = 0$ for all $h \in \mathcal{I}(h_1, \ldots, h_u)$.*

The following theorem is commonly attributed to [124]. See also [18].

Theorem 3.8 (Stengle's Positivstellensatz)**.** *Let $\mathcal{T}$ be the SAS$_{\neq}$ in (3.1) (Definition 3.5). It is infeasible (meaning $[\![\mathcal{T}]\!] = \emptyset$) if and only if there exist $f \in \mathcal{C}(f_1, \ldots, f_s)$, $g \in \mathcal{M}(g_1, \ldots, g_t)$ and $h \in \mathcal{I}(h_1, \ldots, h_u)$ such that $f + g^2 + h = 0$.*

The polynomials f, g, h can be seen as an *infeasible certificate* of the SAS$_{\neq}$ $\mathcal{T}$. The "if" direction is shown easily: if $\vec{x} \in [\![\mathcal{T}]\!]$ then we have $f(\vec{x}) \geq 0$, $g(\vec{x})^2 > 0$ and $h(\vec{x}) = 0$ (by Lemma 3.7), leading to a contradiction. The "only if" direction is nontrivial and remarkable; it is however not used in the algorithm of [39] nor in this chapter.

SOS polynomials play important roles, both theoretically and in the implementation.

Definition 3.9 (sum of squares (SOS)) A polynomial is called a *sum of squares (SOS)* if it can be written in the form $p_1^2 + \cdots + p_N^2$ (for some polynomials $p_1, \ldots, p_N$). Note that $\mathcal{C}(\emptyset)$ is exactly the set of sums of squares (Definition 3.6).

3.2.2 The Interpolation Algorithm by Dai et al.

Definition 3.10 (interpolant) Let $\mathcal{T}$ and $\mathcal{T}'$ be SAS$_{\neq}$'s, in variables $\vec{X}, \vec{Y}$ and in $\vec{X}, \vec{Z}$, respectively, given in the following form. Here we assume that each variable in $\vec{X}$ occurs both in $\mathcal{T}$ and $\mathcal{T}'$, and that $\vec{Y} \cap \vec{Z} = \emptyset$.

$$\begin{aligned} \mathcal{T} &= \left(\begin{array}{ll} f_1(\vec{X},\vec{Y}) \geq 0, \ \ldots, \ f_s(\vec{X},\vec{Y}) \geq 0, & g_1(\vec{X},\vec{Y}) \neq 0, \ \ldots, \ g_t(\vec{X},\vec{Y}) \neq 0, \\ h_1(\vec{X},\vec{Y}) = 0, \ \ldots, \ h_u(\vec{X},\vec{Y}) = 0 & \end{array} \right) \\ \mathcal{T}' &= \left(\begin{array}{ll} f'_1(\vec{X},\vec{Z}) \geq 0, \ \ldots, \ f'_{s'}(\vec{X},\vec{Z}) \geq 0, & g'_1(\vec{X},\vec{Z}) \neq 0, \ \ldots, \ g'_{t'}(\vec{X},\vec{Z}) \neq 0, \\ h'_1(\vec{X},\vec{Z}) = 0, \ \ldots, \ h'_{u'}(\vec{X},\vec{Z}) = 0 & \end{array} \right) \end{aligned} \tag{3.2}$$

Assume further that $\mathcal{T}$ and $\mathcal{T}'$ are *disjoint*, that is, $[\![\mathcal{T}]\!] \cap [\![\mathcal{T}']\!] = \emptyset$.

An SAS$_{\neq}$ $\mathcal{S}$ is an *interpolant* of $\mathcal{T}$ and $\mathcal{T}'$ if it satisfies the following:

1. $[\![\mathcal{T}]\!] \subseteq [\![\mathcal{S}]\!]$;
2. $[\![\mathcal{S}]\!] \cap [\![\mathcal{T}']\!] = \emptyset$; and
3. (the *common variable condition*) the SAS$_{\neq}$ $\mathcal{S}$ is in the variables $\vec{X}$, that is, $\mathcal{S}$ contains only those variables which occur both in $\mathcal{T}$ and $\mathcal{T}'$.

Towards efficient synthesis of nonlinear interpolants, Dai et al. [39] introduced a workflow that hinges on the following variation of the Positivstellensatz.

Theorem 3.11 (disjointness certificate in [39, Section 4])**.** *Let $\mathcal{T}, \mathcal{T}'$ be the SAS$_{\neq}$'s in (3.2). Assume there exist*

$$\begin{array}{l} \widetilde{f} \in \mathcal{C}(f_1, \ldots, f_s, f'_1, \ldots, f'_{s'}) \ , \quad g \in \mathcal{M}(g_1, \ldots, g_t, g'_1, \ldots, g'_{t'}) \quad \textit{and} \\ \widetilde{h} \in \mathcal{I}(h_1, \ldots, h_u, h'_1, \ldots, h'_{u'}) \ , \quad \textit{such that} \quad 1 + \widetilde{f} + g^2 + \widetilde{h} \ = \ 0 \ . \end{array} \tag{3.3}$$

Assume further that $\widetilde{f}$ allows a decomposition $\widetilde{f} = f + f'$, with some $f \in \mathcal{C}(f_1, \ldots, f_s)$ and $f' \in \mathcal{C}(f'_1, \ldots, f'_{s'})$. (An element $\widetilde{h}$ in the ideal always allows a decomposition $\widetilde{h} = h + h'$ such that $h \in \mathcal{I}(h_1, \ldots, h_u)$ and $h' \in \mathcal{I}(h'_1, \ldots, h'_{u'})$.)

Under the assumptions $\mathcal{T}$ and $\mathcal{T}'$ are disjoint. Moreover the SAS$_{\neq}$

$$\mathcal{S} \ \coloneqq \ \big(1/2 + f + g^2 + h > 0 \big) \tag{3.4}$$

satisfies the conditions of an interpolant of $\mathcal{T}$ and $\mathcal{T}'$ (Definition 3.10), except for Cond. 3. (the common variable condition).

Algorithm 1 The interpolation algorithm by Dai et al. [39]. Here $\mathbf{2} = \{0, 1\}$

1: **input**: SAS$_{\neq}$'s $\mathcal{T}, \mathcal{T}'$ in (3.2), and $b \in \mathbb{N}$ (the maximum degree)

2: **output**: either an interpolant $\mathcal{S}$ of $\mathcal{T}$ and $\mathcal{T}'$, or FAIL

3: $\widetilde{g} \coloneqq (\prod_{i=1}^{t} g_i)(\prod_{i'=1}^{t'} g'_{i'})$; $g \coloneqq \widetilde{g}^{\lfloor b/2 \deg(h) \rfloor}$ ▷ g is roughly of degree $b/2$

4: Solve PDioph$^{\mathrm{SOS}}$ to find $(\overrightarrow{\alpha}, \overrightarrow{\alpha'}, \overrightarrow{\beta}, \overrightarrow{\beta'})$. Here:

- $\alpha_i \in \mathcal{C}(\emptyset)_{\leq b}$ (for $i \in \mathbf{2}^s$) and $\alpha'_{i'} \in \mathcal{C}(\emptyset)_{\leq b}$ (for $i' \in \mathbf{2}^{s'}$) are SOSs,
- $\beta_j \in \mathbb{R}[\vec{X}]_{\leq b}$ (for $j \in [1, u]$) and $\beta'_{j'} \in \mathbb{R}[\vec{X}]_{\leq b}$ (for $j' \in [1, u']$) are polynomials,
- and they are subject to the constraint

$$1 + \sum_{i \in \mathbf{2}^s} \alpha_i f_1^{i_1} \cdots f_s^{i_s} + \sum_{i' \in \mathbf{2}^{s'}} \alpha'_{i'} {f'}_1^{i'_1} \cdots {f'}_{s'}^{i'_{s'}} + g^2 + \sum_{j=1}^{u} \beta_j h_j + \sum_{j'=1}^{u'} \beta'_{j'} h'_{j'} = 0 \ . \quad (3.6)$$

(Such $(\overrightarrow{\alpha}, \overrightarrow{\alpha'}, \overrightarrow{\beta}, \overrightarrow{\beta'})$ may not be found, in which case return FAIL)

5: $f \coloneqq \sum_{i \in \mathbf{2}^s} \alpha_i f_1^{i_1} \cdots f_s^{i_s}$; $h \coloneqq \sum_{j=1}^{u} \beta_j h_j$

6: **return** $\mathcal{S} \coloneqq \big(1/2 + f + g^2 + h > 0\big)$

Proof. The proof is much like the "if" part of Theorem 3.8. It suffices to show that $\mathcal{S}$ is an interpolant; then the disjointness of $\mathcal{T}$ and $\mathcal{T}'$ follows.

To see $[\![\mathcal{T}]\!] \subseteq [\![\mathcal{S}]\!]$, assume $\vec{x} \in [\![\mathcal{T}]\!]$. Then we have $f(\vec{x}) \geq 0$ and $h(\vec{x}) = 0$ by Lemma 3.7; additionally $\big(g(\vec{x})\big)^2 \geq 0$ holds too. Thus $1/2 + f(\vec{x}) + \big(g(\vec{x})\big)^2 + h(\vec{x}) \geq 1/2 > 0$ and we have $\vec{x} \in [\![\mathcal{S}]\!]$.

To see $[\![\mathcal{S}]\!] \cap [\![\mathcal{T}']\!] = \emptyset$, we firstly observe that the following holds for any $\vec{x}$.

$$\begin{array}{rcl} 0 & = & 1 + f(\vec{x}) + f'(\vec{x}) + \big(g(\vec{x})\big)^2 + h(\vec{x}) + h'(\vec{x}) \quad \text{by (3.3)} \\ & = & \big(\,1/2 + f(\vec{x}) + (g(\vec{x}))^2 + h(\vec{x})\,\big) + \big(\,1/2 + f'(\vec{x}) + h'(\vec{x})\,\big) \ . \end{array} \quad (3.5)$$

Assume $\vec{x} \in [\![\mathcal{S}]\!] \cap [\![\mathcal{T}']\!]$. By $\vec{x} \in [\![\mathcal{S}]\!]$ we have $1/2 + f(\vec{x}) + (g(\vec{x}))^2 + h(\vec{x}) > 0$; and by $\vec{x} \in [\![\mathcal{T}']\!]$ we have $f'(\vec{x}) \geq 0$ and $h'(\vec{x}) = 0$ (Lemma 3.7), hence $1/2 + f'(\vec{x}) + h'(\vec{x}) \geq 1/2 > 0$. Thus the right-hand side of (3.5) is strictly positive, a contradiction. ■

Note that we no longer have completeness: existence of an interpolant like (3.4) is not guaranteed. Nevertheless Theorem 3.11 offers a sound method to construct an interpolant, namely by finding a suitable disjointness certificate f, f', g, h, h'.

The interpolation algorithm in [39] is shown in Algorithm 1, where the search for a disjointness certificate f, f', g, h, h' is relaxed to the following problem.

Definition 3.12 (PDioph$^{\mathrm{SOS}}$) Let PDioph$^{\mathrm{SOS}}$ stand for the following problem.

Input:	polynomials $\varphi_1, \ldots, \varphi_n,\ \psi_1, \ldots, \psi_m,\ \xi\ \in \mathbb{R}[\vec{X}]$, and maximum degrees $d_1, \ldots, d_n, e_1, \ldots, e_m \in \mathbb{N}$
Output:	SOSs $s_1 \in \mathcal{C}(\emptyset)_{\leq d_1}, \ldots, s_n \in \mathcal{C}(\emptyset)_{\leq d_n}$ and polynomials $t_1 \in \mathbb{R}[\vec{X}]_{\leq e_1}, \ldots, t_m \in \mathbb{R}[\vec{X}]_{\leq e_m}$ such that $s_1\varphi_1 + \cdots + s_n\varphi_n + t_1\psi_1 + \cdots + t_m\psi_m + \xi = 0$

Here $\mathbb{R}[\vec{X}]_{\leq e}$ denotes the set of polynomials in $\vec{X}$ whose degree is no bigger than e. Similarly $\mathcal{C}(\emptyset)_{\leq d}$ is the set of SOSs with degree $\leq d$.

The problem PDioph$^{\mathrm{SOS}}$ is principally about finding polynomials s_i, t_j subject to $\sum_i s_i\varphi_i + \sum_j t_j\psi_j + \xi = 0$; this problem is known as *polynomial Diophantine equations*. In PDioph$^{\mathrm{SOS}}$ SOS requirements are additionally imposed on part of a solution (namely s_i); degrees are bounded, too, for algorithmic purposes.

In Algorithm 1 we rely on Theorem 3.11 to generate an interpolant: roughly speaking, one looks for a disjointness certificate f, f', g, h, h' within a predetermined maximum degree b. This search is relaxed to an instance of PDioph$^{\mathrm{SOS}}$ (Definition 3.12), with $n = 2^s + 2^{s'}$, $m = u + u'$, and $\xi = 1 + g^2$, as in Line 4. The last relaxation, introduced in [39], is derived from the following representation of elements of the cone $\mathcal{C}(\overrightarrow{f}, \overrightarrow{f'})$, the multiplicative monoid $\mathcal{M}(\overrightarrow{g}, \overrightarrow{g'})$ and the ideal $\mathcal{I}(\overrightarrow{h}, \overrightarrow{h'})$, respectively.

- Each element h of $\mathcal{I}(\overrightarrow{h}, \overrightarrow{h'})$ is of the form $h = \sum_{j=1}^{u} \beta_j h_j + \sum_{j'=1}^{u'} \beta'_{j'} h'_{j'}$, where $\beta_j, \beta_{j'} \in \mathbb{R}[\vec{X}]$. This is a standard fact in ring theory.
- Each element of $\mathcal{M}(\overrightarrow{g}, \overrightarrow{g'})$ is given by the product of finitely many elements from $\overrightarrow{g}, \overrightarrow{g'}$ (here multiplicity matters). In Algorithm 1 a polynomial g is fixed to a "big" one. This is justified as follows: in case the constraint (3.6) is satisfiable using a smaller polynomial g' instead of g, by multiplying the whole equality (3.6) by $1 + (g/g')^2$ we see that (3.6) is satisfiable using g, too.
- For the cone $\mathcal{C}(\overrightarrow{f}, \overrightarrow{f'})$ we use the following fact (here $\mathbf{2} = \{0, 1\}$). The lemma seems to be widely known but we present a proof for the record.

Lemma 3.13. *An arbitrary element f of the cone $\mathcal{C}(f_1, \ldots, f_s)$ can be expressed as $f = \sum_{i \in \mathbf{2}^s} \alpha_i f_1^{i_1} \ldots f_s^{i_s}$, using SOSs α_i (where $i \in \mathbf{2}^s$).*

We need to define *quadratic module* to prove this lemma.

Definition 3.14 (quadratic module) A *quadratic module* generated by $M = (m_\lambda)_{\lambda \in \Lambda} \in \mathbb{R}[\vec{X}]^\Lambda$

(Λ is an index set) is the set

$$\mathcal{QM}(M) \coloneqq \left\{ q + \sum_{\lambda\in\Lambda} q_\lambda m_\lambda \;\middle|\; \begin{array}{l} q \in \mathcal{C}(\emptyset),\ (q_\lambda)_{\lambda\in\Lambda} \in \mathcal{C}(\emptyset)^\Lambda, \text{ and} \\ q_\lambda = 0 \text{ except for finitely many } \lambda\text{'s} \end{array} \right\}. \tag{3.7}$$

Lemma 3.13 is an immediate consequence of the following lemma.

Lemma 3.15 (Proof of Lemma 3.13)**.** *Let $f_1, \ldots, f_s \in \mathbb{R}[\vec{X}]$. Then,*

$$\mathcal{C}(f_1, \ldots, f_s) = \mathcal{QM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbf{2}^s) \tag{3.8}$$

where $i = (i_1, \ldots, i_s)$.

Proof. The left-to-right direction is proved by induction on the construction of $\mathcal{C}(f_1, \ldots, f_s)$. Let $f \in \mathcal{C}(f_1, \ldots, f_s)$.

1. *Case: $f = f_\lambda$ for some $\lambda = 1, \ldots, s$.*

$$f = f_\lambda = f_1^0 \cdots f_\lambda^1 \cdots f_s^0 \in \mathcal{QM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbf{2}^s). \tag{3.9}$$

2. *Case: $f = g^2$ for some $g \in \mathbb{R}[\vec{X}]$.*

$$f = g^2 \in \mathcal{C}(\emptyset) \subset \mathcal{QM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbf{2}^s). \tag{3.10}$$

3. *Case: $f = g + h$ for some $g, h \in \mathcal{C}(f_1, \ldots, f_s) \cap \mathcal{QM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbf{2}^s)$.* Obvious, because $\mathcal{QM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbf{2}^s)$ is closed under addition.
4. *Case: $f = gh$ for some $g, h \in \mathcal{C}(f_1, \ldots, f_s) \cap \mathcal{QM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbf{2}^s)$.* g and h have the form

$$g = \sum_{i\in\mathbf{2}^s} q_i f_1^{i_1} \cdots f_s^{i_s}, \quad h = \sum_{j\in\mathbf{2}^s} r_j f_1^{j_1} \cdots f_s^{j_s}, \tag{3.11}$$

where $q_i, r_j \in \mathcal{C}(\emptyset)$. Let $p_\lambda = \sum_{i+j=\lambda} q_i r_j \in \mathcal{C}(\emptyset)$. Then

$$\begin{aligned} gh &= \sum_{i\in\mathbf{2}^s} \sum_{j\in\mathbf{2}^s} q_i r_j f_1^{i_1+j_1} \cdots f_s^{i_s+j_s} && (3.12)\\ &= \sum_{\lambda\in\mathbf{2}^s} \sum_{i+j=\lambda} q_i r_j f_1^{\lambda_1} \cdots f_s^{\lambda_s} && (3.13)\\ &= \sum_{\lambda\in\mathbf{2}^s} p_\lambda f_1^{\lambda_1} \cdots f_s^{\lambda_s} && (3.14)\\ &= \sum_{\lambda\in\mathbf{2}^s} \left(p_\lambda f_1^{2\lfloor \lambda_1/2 \rfloor} \cdots f_s^{2\lfloor \lambda_s/2 \rfloor} \right) f_1^{\lambda_1 \bmod 2} \cdots f_s^{\lambda_s \bmod 2}. && (3.15) \end{aligned}$$

Because $p_\lambda f_1^{2\lfloor \lambda_1/2 \rfloor} \cdots f_s^{2\lfloor \lambda_s/2 \rfloor} \in \mathcal{C}(\emptyset)$ for each $\lambda \in 2^s$, we have $f = gh \in \mathcal{QM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbf{2}^s)$.

For the converse, let $f \in \mathcal{QM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbf{2}^s)$. Then f has the form $f = \sum_{i \in \mathbf{2}^s} q_i f_1^{i_1} \cdots f_s^{i_s}$ where $q_i \in \mathcal{C}(\emptyset) \subseteq \mathcal{C}(f_1, \ldots, f_s)$. Then f indeed belongs to $\mathcal{C}(f_1, \ldots, f_s)$, because cones are closed under addition and multiplication. ■

The last representation in Lemma 3.13 justifies the definition of f and h in Algorithm 1 (Line 5). We also observe that Line 6 of Algorithm 1 corresponds to (3.4) of Theorem 3.11.

In implementing Algorithm 1 the following fact is crucial (see [39, Section 3.5] and also [106, 107] for details): the problem $\text{PDioph}^{\text{SOS}}$ (Definition 3.12) can be reduced to an SDP problem, the latter allowing an efficient solution by state-of-the-art SDP solvers. It should be noted, however, that numerical errors (inevitable in interior point methods) can pose a serious issue for our application: the constraint (3.6) is an equality and hence fragile.

3.3 Positivstellensatz and Interpolation, Revisited

3.3.1 Analysis of the Interpolation Algorithm by Dai et al.

Intrigued by its solid mathematical foundation in real algebraic geometry as well as its efficient implementation that exploits state-of-the-art SDP solvers, we studied the framework by Dai et al. [39] (it was sketched in Section 3.2.2). In its course we obtained the following observations that motivate our current technical contributions.

We first observed that Algorithm 1 from [39] fails to find "sharp" interpolants for "barely disjoint" predicates (see Example 3.1). This turns out to be a general phenomenon (see Prop. 3.18).

Definition 3.16 (symbolic closure) Let $\mathcal{T}$ be the $\text{SAS}_{\neq}$'s in (3.1). The *symbolic closure* $\mathcal{T}_\bullet$ of $\mathcal{T}$ is the $\text{SAS}_{\neq}$ that is obtained by dropping all the disequality constraints $g_j(\vec{x}) \neq 0$ in $\mathcal{T}$.

$$\mathcal{T}_\bullet = \big(\; f_1(\vec{X}, \vec{Y}) \geq 0 \;,\; \ldots,\; f_s(\vec{X}, \vec{Y}) \geq 0 \;,\; h_1(\vec{X}, \vec{Y}) = 0 \;,\; \ldots,\; h_u(\vec{X}, \vec{Y}) = 0 \; \big) \qquad (3.16)$$

The intuition of symbolic closure of $\mathcal{T}$ is to replace all strict inequalities $g'_j(\vec{X}, \vec{Y}) > 0$ in $\mathcal{T}$ with the corresponding non-strict ones $g'_j(\vec{X}, \vec{Y}) \geq 0$. Since only $\geq, \neq$ and $=$ are allowed in $\text{SAS}_{\neq}$'s, strict inequalities $g'_j(\vec{X}, \vec{Y}) > 0$ are presented in the $\text{SAS}_{\neq}$ $\mathcal{T}$ by using both $g'_j(\vec{X}, \vec{Y}) \geq 0$ and $g'_j(\vec{X}, \vec{Y}) \neq 0$. The last definition drops the latter disequality ($\neq$) requirement.

The notion of symbolic closure most of the time coincides with closure with respect to the usual Euclidean topology, but not in some singular cases. See Appendix 3.3.2 for details.

Definition 3.17 (bare disjointness) Let $\mathcal{T}$ and $\mathcal{T}'$ be $\text{SAS}_{\neq}$'s. $\mathcal{T}$ and $\mathcal{T}'$ are *barely disjoint* if $[\![\mathcal{T}]\!] \cap [\![\mathcal{T}']\!] = \emptyset$ and $[\![\mathcal{T}_\bullet]\!] \cap [\![\mathcal{T}'_\bullet]\!] \neq \emptyset$.

An interpolant $\mathcal{S}$ of barely disjoint $\text{SAS}_{\neq}$'s $\mathcal{T}$ and $\mathcal{T}'$ shall be said to be *sharp*.

An example of barely disjoint SAS$_{\neq}$'s is in Example 3.1: $(0,0) \in [\![\mathcal{T}_\bullet]\!] \cap [\![\mathcal{T}'_\bullet]\!] \neq \emptyset$.

Algorithm 1 does not work if the SAS$_{\neq}$'s $\mathcal{T}$ and $\mathcal{T}'$ are only barely disjoint. In fact, such a failure is theoretically guaranteed, as the following result states. Its proof is very similar to the proof of Theorem 3.11.

Proposition 3.18. *Let $\mathcal{T}$ and $\mathcal{T}'$ be the SAS$_{\neq}$'s in (3.2). If $\mathcal{T}$ and $\mathcal{T}'$ are barely disjoint (in the sense of Definition 3.17), there do not exist polynomials $\widetilde{f} \in \mathcal{C}(\overrightarrow{f}, \overrightarrow{f'})$, $g \in \mathcal{M}(\overrightarrow{g}, \overrightarrow{g'})$ and $\widetilde{h} \in \mathcal{I}(\overrightarrow{h}, \overrightarrow{h'})$ such that $1 + \widetilde{f} + g^2 + \widetilde{h} = 0$.*

Proof. We argue by contradiction. Assume that there exist polynomials $\widetilde{f}, g, \widetilde{h}$ that satisfy the conditions in the proposition. By the feasibility assumption there exists some $\vec{r} \in [\![\mathcal{T}_\bullet]\!] \cap [\![\mathcal{T}'_\bullet]\!]$. For this $\vec{r}$ we have $\widetilde{f}(\vec{r}) + \widetilde{h}(\vec{r}) \geq 0$ by Lemma 3.7, while by $1 + \widetilde{f} + g^2 + \widetilde{h} = 0$ we have $\widetilde{f}(\vec{r}) + \widetilde{h}(\vec{r}) = -1 - \big(g(\vec{r})\big)^2 < 0$. Contradiction. ■

The conditions in Prop. 3.18 on the polynomials $\widetilde{f}, g, \widetilde{h}$ are those for disjointness certificates for $\mathcal{T}$ and $\mathcal{T}'$ (Theorem 3.11). As a consequence: if $\mathcal{T}$ and $\mathcal{T}'$ are only barely disjoint, interpolation relying on Theorem 3.11—that underlies the framework in [39]—never succeeds.

3.3.2 Topological and Algebraic Closure

In this section, let us consider the difference between topological closure and algebraic closure, that is, the difference between $[\![\mathcal{T}_\bullet]\!] \subseteq \mathbb{R}^k$ (where $\mathcal{T}_\bullet$ is from Definition 3.16) and $\overline{[\![\mathcal{T}]\!]} \subseteq \mathbb{R}^k$. Here $\overline{(\_)}$ refers to the closure with respect to the Euclidean topology of $\mathbb{R}^k$. Readers can safely skip over this section.

Those closures coincide in many cases but not always do so. For example, for $\mathcal{T} = (x^3 - 2x^2 + x \leq 0, x^3 - 2x^2 + x \neq 0)$, we have $[\![\mathcal{T}_\bullet]\!] = [\![x^3 - 2x^2 + x \leq 0]\!] = (-\infty, 0] \cup \{\,1\,\}$, but $\overline{[\![\mathcal{T}]\!]} = \overline{(-\infty, 0)} = (-\infty, 0]$.

We can show the following inclusion in general.

Proposition 3.19. *Let $n \geq 1$ and $\mathcal{A} = (f_1 \triangleright_1 0, \ldots, f_n \triangleright_n 0)$ be an SAS$_{\neq}$, where $\triangleright_i \in \{\geq, \neq, =\}$. Then*

$$\overline{[\![\mathcal{A}]\!]} \subseteq [\![\mathcal{A}_\bullet]\!]. \tag{3.17}$$

Proof. The proof is by induction on n.

- Base cases: The cases $\triangleright_1 = (=)$ and $\triangleright_1 = (\geq)$ are easy because f_1 is continuous and both $\{\,0\,\}$ and $[0, \infty)$ are closed in $\mathbb{R}$. For the remaining case where $\triangleright_1 = (\neq)$, we have $\overline{[\![f_1 \neq 0]\!]} \subseteq \mathbb{R}^k = [\![()]\!] = [\![\mathcal{A}_\bullet]\!]$.

- Step case: Let $\mathcal{B} = (f_1 \triangleright_1 0, \ldots, f_n \triangleright_n 0)$.

$$\begin{aligned}
\overline{[\![\mathcal{A}]\!]} &= \overline{[\![\mathcal{B}, f_{n+1} \triangleright_{n+1} 0]\!]} \\
&= \overline{[\![\mathcal{B}]\!]} \cap \overline{[\![f_{n+1} \triangleright_{n+1} 0]\!]} \\
&\subseteq \overline{[\![\mathcal{B}]\!]} \cap \overline{[\![f_{n+1} \triangleright_{n+1} 0]\!]} \\
&\subseteq [\![\mathcal{B}_\bullet]\!] \cap [\![(f_{n+1} \triangleright_{n+1} 0)_\bullet]\!] \\
&\qquad \text{(by the induction hypothesis and arguments similar to the base case)} \\
&= [\![\mathcal{A}_\bullet]\!].
\end{aligned}$$

■

It follows that $\overline{[\![\mathcal{T}]\!]} \cap \overline{[\![\mathcal{T}']\!]} \subseteq [\![\mathcal{T}_\bullet]\!] \cap [\![\mathcal{T}'_\bullet]\!]$. The opposite inclusion fails in general: for $\mathcal{T} = (x^3 - 2x^2 + x \leq 0, x^3 - 2x^2 + x \neq 0), \mathcal{T}' = (x \geq 0)$, $\overline{[\![\mathcal{T}]\!]} \cap \overline{[\![\mathcal{T}']\!]} = (-\infty, 0) \cap [0, \infty) = \emptyset$ and $[\![\mathcal{T}_\bullet]\!] \cap [\![\mathcal{T}'_\bullet]\!] = ((-\infty, 0) \cup \{\, 1 \,\}) \cap [0, \infty) = \{\, 1 \,\}$.

3.3.3 Interpolation via Positivstellensatz, Sharpened

The last observation motivates our "sharper" variant of Theorem 3.11—a technical contribution that we shall present shortly in Theorem 3.24. We switch input formats by replacing disequalities $\neq$ (Definition 3.5) with $<$. This small change turns out to be useful when we formulate our main result (Theorem 3.24).

Definition 3.20 ($\mathrm{SAS}_<$) A *semialgebraic system with strict inequalities ($\mathrm{SAS}_<$)* $\mathcal{T}$, in variables $X_1, X_2, \ldots, X_k$, is a sequence

$$\mathcal{T} \;=\; \left(\begin{array}{l} f_1(\vec{X}) \geq 0 \;,\; \ldots,\; f_s(\vec{X}) \geq 0 \;, \\ h_1(\vec{X}) = 0 \;,\; \ldots,\; h_u(\vec{X}) = 0 \end{array} \qquad g_1(\vec{X}) > 0 \;,\; \ldots,\; g_t(\vec{X}) > 0 \;, \right) \tag{3.18}$$

of inequalities $f_i(\vec{X}) \geq 0$, *strict inequalities* $g_j(\vec{X}) > 0$ and equalities $h_k(\vec{X}) = 0$. Here $f_i, g_j, h_k \in \mathbb{R}[\vec{X}]$ are polynomials; $[\![\mathcal{T}]\!] \subseteq \mathbb{R}^k$ is defined like in Definition 3.5.

$\mathrm{SAS}_<$'s have the same expressive power as $\mathrm{SAS}_\neq$'s, as witnessed by the following mutual translation. For the $\mathrm{SAS}_\neq$ $\mathcal{T}$ in (3.1), the $\mathrm{SAS}_<$ $\widetilde{\mathcal{T}} := \big(f_i(\vec{X}) \geq 0, g_j^2(\vec{X}) > 0, h_k(\vec{X}) = 0 \big)_{i,j,k}$ satisfies $[\![\mathcal{T}]\!] = [\![\widetilde{\mathcal{T}}]\!]$. Conversely, for the $\mathrm{SAS}_<$ $\mathcal{T}$ in (3.18), the $\mathrm{SAS}_\neq$ $\widehat{\mathcal{T}} := \big(f_i(\vec{X}) \geq 0, g_j(\vec{X}) \geq 0, g_j(\vec{X}) \neq 0, h_k(\vec{X}) = 0 \big)_{i,j,k}$ satisfies $[\![\mathcal{T}]\!] = [\![\widehat{\mathcal{T}}]\!]$.

One crucial piece for the Positivstellensatz was the closure properties of inequalities/disequalities/equalities encapsulated in the notions of cone/multiplicative monoid/ideal (Lemma 3.7). We devise a counterpart for strict inequalities.

Definition 3.21 (strict cone) A set $S \subseteq \mathbb{R}[\vec{X}]$ is a *strict cone* if it satisfies the following closure properties: 1) $f, g \in S$ implies $f + g \in S$; 2) $f, g \in S$ implies $fg \in S$; and 3) $r \in S$

for any positive real $r \in \mathbb{R}_{>0}$. For a subset A of $\mathbb{R}[\vec{X}]$, we write $\mathcal{SC}(A)$ for the smallest strict cone that includes A.

Lemma 3.22. *Let $\vec{x} \in \mathbb{R}^k$ and $g_j \in \mathbb{R}[\vec{X}]$. If $\vec{x} \in [\![g_1 > 0, \ldots, g_t > 0]\!]$, then $g(\vec{x}) > 0$ for all $g \in \mathcal{SC}(g_1, \ldots, g_t)$.*

We can now formulate adaptation of Positivstellensatz.

Theorem 3.23 (Positivstellensatz for $\mathrm{SAS}_<$)**.** *Let $\mathcal{T}$ be the $SAS_<$ in (3.18). It is infeasible (i.e. $[\![\mathcal{T}]\!] = \emptyset$) if and only if there exist $f \in \mathcal{C}(f_1, \ldots, f_s, g_1, \ldots, g_t)$, $g \in \mathcal{SC}(g_1, \ldots, g_t)$ and $h \in \mathcal{I}(h_1, \ldots, h_u)$ such that $f + g + h = 0$.*

Proof. The "if" direction is easy. Suppose there exist such polynomials f, g, h as required. Assume $\vec{r} \in [\![\mathcal{T}]\!]$; then we have $f(\vec{r}) \geq 0$, $g(\vec{r}) > 0$, and $h(\vec{r}) = 0$ by Lemma 3.7 and 3.22. Thus we have $f(\vec{r}) + g(\vec{r}) + h(\vec{r}) > 0$, which contradicts with $f + g + h = 0$. Therefore $[\![\mathcal{T}]\!] = \emptyset$.

For the "only if" direction we rely on Theorem 3.8, via the translation of $\mathrm{SAS}_<$'s to $\mathrm{SAS}_{\neq}$'s that we described after Definition 3.20, and use the fact that $\mathcal{M}(g_1, \ldots, g_t) \subseteq \mathcal{SC}(g_1, \ldots, g_t)$. ■

From this we derive the following adaptation of Theorem 3.11 that allows to synthesize sharp interpolants. The idea is as follows. In Theorem 3.11, the constants 1 (in (3.3)) and 1/2 (in (3.4)) are there to enforce strict positivity. This is a useful trick but sometimes too "dull": one can get rid of these constants and still make the proof of Theorem 3.11 work, for example when $g(\vec{x})$ happens to belong to $\mathcal{SC}(g_1, \ldots, g_t)$ instead of $\mathcal{M}(g_1, \ldots, g_t, g'_1, \ldots, g'_{t'})$.

Theorem 3.24 (disjointness certificate from strict cones)**.** *Let $\mathcal{T}$ and $\mathcal{T}'$ be the following $SAS_<$'s, where $\vec{X}$ denotes the variables that occur in both of $\mathcal{T}, \mathcal{T}'$.*

$$\mathcal{T} = \left(\begin{array}{l} f_1(\vec{X}, \vec{Y}) \geq 0 \ , \ \ldots, \ f_s(\vec{X}, \vec{Y}) \geq 0 \ , \quad g_1(\vec{X}, \vec{Y}) > 0 \ , \ \ldots, \ g_t(\vec{X}, \vec{Y}) > 0 \ , \\ h_1(\vec{X}, \vec{Y}) = 0 \ , \ \ldots, \ h_u(\vec{X}, \vec{Y}) = 0 \end{array} \right),$$
$$\mathcal{T}' = \left(\begin{array}{l} f'_1(\vec{X}, \vec{Z}) \geq 0 \ , \ \ldots, \ f'_{s'}(\vec{X}, \vec{Z}) \geq 0 \ , \quad g'_1(\vec{X}, \vec{Z}) > 0 \ , \ \ldots, \ g'_{t'}(\vec{X}, \vec{Z}) > 0 \ , \\ h'_1(\vec{X}, \vec{Z}) = 0 \ , \ \ldots, \ h'_{u'}(\vec{X}, \vec{Z}) = 0 \end{array} \right). \tag{3.19}$$

Assume there exist

$$\begin{array}{l} f \in \mathcal{C}(f_1, \ldots, f_s, g_1, \ldots, g_t) \ , \quad f' \in \mathcal{C}(f'_1, \ldots, f'_{s'}, g'_1, \ldots, g'_{t'}) \ , \\ g \in \mathcal{SC}(g_1, \ldots, g_t) \ , \quad h \in \mathcal{I}(h_1, \ldots, h_u) \ , \quad \text{and} \quad h' \in \mathcal{I}(h'_1, \ldots, h'_{u'}) \\ \quad \text{such that} \quad f + f' + g + h + h' \ = \ 0 \ . \end{array} \tag{3.20}$$

Then the $SAS_<$'s $\mathcal{T}$ and $\mathcal{T}'$ are disjoint. Moreover the $SAS_<$

$$\mathcal{S} \ \coloneqq \ (f + g + h > 0) \tag{3.21}$$

satisfies the conditions of an interpolant of $\mathcal{T}$ and $\mathcal{T}'$ (Definition 3.10), except for Cond. 3. (the common variable condition).

The proof is similar to the proof of Theorem 3.11. We also have the following symmetric variant.

Theorem 3.25. *Assume the conditions of Theorem 3.24, but let us now require $g \in \mathcal{SC}(g'_1, \dots, g'_{t'})$ (instead of $g \in \mathcal{SC}(g_1, \dots, g_t)$). Then $\mathcal{S} = (f + h \geq 0)$ is an interpolant of $\mathcal{T}$ and $\mathcal{T}'$ (except for the common variable condition).*

Example 3.26 Let us apply Theorem 3.24 to $\mathcal{T} = (-y > 0)$ and $\mathcal{T}' = (y - x \geq 0, y + x \geq 0)$ (these are only barely disjoint). There exists a disjointness certificate f, f', g, h, h': indeed, we can take $f = 0 \in \mathcal{C}(-y)$, $f' = 2y = (y - x) + (y + x) \in \mathcal{C}(y - x, y + x)$, $g = 2(-y) \in \mathcal{SC}(-y)$, and $h = h' = 0 \in \mathcal{I}(\emptyset)$; for these we have $f + f' + g + h + h' = 0$. This way an interpolant $\mathcal{S} = (f + g + h > 0) = (-2y > 0)$ is derived.

Remark 3.27 Our use of strict cones allows to use a polynomial g in (3.21). This is in contrast with g^2 in (3.4) and yields an interpolant of a potentially smaller degree.

We derive an interpolation algorithm from Theorem 3.24; see Algorithm 2. An algorithm based on Theorem 3.25 can be derived similarly, too.

Algorithm 2 the reduces search for a disjointness certificate f, f', g, h, h' (from Theorem 3.24) to a problem similar to $\text{PDioph}^{\text{SOS}}$ (Line 3). Unlike the original definition of $\text{PDioph}^{\text{SOS}}$ (Definition 3.12), here we impose additional constraints (3.23–3.24) other than the equality (3.22) that comes from (3.20). It turns out that, much like $\text{PDioph}^{\text{SOS}}$ allows relaxation to SDP problems [39, 106, 107], the problem in Line 3 can also be reduced to an SDP problem.

The constraint (3.23) is there to force $g = \sum_{k \in \mathbf{b}^t} \gamma_k g_1^{k_1} \cdots g_t^{k_t}$ (see (4)) to belong to the *strict* cone $\mathcal{SC}(g_1, \dots, g_t)$. A natural requirement $\sum_{k \in \mathbf{b}^t} \gamma_k > 0$ for that purpose does not allow encoding to an SDP constraint so we do not use it. Our relaxation from $\sum_{k \in \mathbf{b}^t} \gamma_k > 0$ to $\sum_{k \in \mathbf{b}^t} \gamma_k \geq 1$ is inspired by [116]; it does not lead to a loss of generality in our current task of finding polynomial certificates.

The constraints (3.24) are extracted in the following straightforward manner: we look at the coefficient of each monomial in $f + g + h$ (see Line 5); and for each monomial that involves variables other than $\vec{X}$ we require the coefficient to be equal to 0. The constraint is linear in the SDP variables, that we can roughly consider as the coefficients of the monomials in $\overrightarrow{\alpha}, \overrightarrow{\alpha'}, \overrightarrow{\beta}, \overrightarrow{\beta'}, \overrightarrow{\gamma}$.

Derivation of Algorithm 2 from Theorem 3.25 also relies on the following analogue of Lemma 3.13.

Algorithm 2 Our interpolation algorithm based on Theorem 3.24. Here $\mathbf{2} = \{0, 1\}$ and $\sigma(b) = \{(k_1, \ldots, k_t) \in \mathbb{N}^t \mid k_1 + \cdots + k_t \leq b + 1\}$

1: **input**: $\mathrm{SAS}_<$'s $\mathcal{T}, \mathcal{T}'$ in (3.19), and $b \in \mathbb{N}$ (the maximum degree)

2: **output**: either an interpolant $\mathcal{S}$ of $\mathcal{T}$ and $\mathcal{T}'$, or FAIL

3: Solve (an extension of) $\mathrm{PDioph}^{\mathrm{SOS}}$ to find $(\overrightarrow{\alpha}, \overrightarrow{\alpha'}, \overrightarrow{\beta}, \overrightarrow{\beta'}, \overrightarrow{\gamma})$. Here:

- $\alpha_{ij} \in \mathcal{C}(\emptyset)_{\leq b}$ (for $i \in \mathbf{2}^s$, $j \in \mathbf{2}^t$) and $\alpha'_{i',j'} \in \mathcal{C}(\emptyset)_{\leq b}$ (for $i' \in \mathbf{2}^{s'}$, $j' \in \mathbf{2}^{t'}$) are SOSs,
- $\beta_j \in \mathbb{R}[\vec{X}]_{\leq b}$ (for $j \in [1, u]$) and $\beta'_{j'} \in \mathbb{R}[\vec{X}]_{\leq b}$ (for $j' \in [1, u']$) are polynomials,
- and $\gamma_k \in \mathbb{R}_{\geq 0}$ (for $k \in \sigma(b)$) are nonnegative real numbers,

that are subject to the constraints

$$\begin{array}{l} \sum_{i \in \mathbf{2}^s,\, j \in \mathbf{2}^t} \alpha_{ij} f_1^{i_1} \cdots f_s^{i_s} g_1^{j_1} \cdots g_t^{j_t} \\ + \sum_{i' \in \mathbf{2}^{s'},\, j' \in \mathbf{2}^{t'}} \alpha'_{i'j'} {f'}_1^{i'_1} \cdots {f'}_{s'}^{i'_{s'}} {g'}_1^{j'_1} \cdots {g'}_{t'}^{j'_{t'}} \\ + \sum_{k \in \sigma(b)} \gamma_k g_1^{k_1} \cdots g_t^{k_t} \;+\; \sum_{j=1}^{u} \beta_j h_j \;+\; \sum_{j'=1}^{u'} \beta'_{j'} h'_{j'} \quad = \quad 0 \;, \end{array} \tag{3.22}$$

$$\sum_{k \in \sigma(b)} \gamma_k \geq 1 \;, \quad \text{and} \tag{3.23}$$

$$\text{some equality constraints that forces the common variable condition.} \tag{3.24}$$

(Such $(\overrightarrow{\alpha}, \overrightarrow{\alpha'}, \overrightarrow{\beta}, \overrightarrow{\beta'}, \overrightarrow{\gamma})$ may not be found, in which case return FAIL)

4: $f \coloneqq \sum_{i \in \mathbf{2}^s,\, j \in \mathbf{2}^t} \alpha_{ij} f_1^{i_1} \cdots f_s^{i_s} g_1^{j_1} \cdots g_t^{j_t}$; $g \coloneqq \sum_{k \in \sigma(b)} \gamma_k g_1^{k_1} \cdots g_t^{k_t}$; $h \coloneqq \sum_{j=1}^{u} \beta_j h_j$

5: **return** $\mathcal{S} \coloneqq (f + g + h > 0)$

Lemma 3.28. *An arbitrary element of the strict cone $\mathcal{SC}(f_1, \ldots, f_s)$ can be expressed as $\sum_{i \in \mathbb{N}^s} \alpha_i f_1^{i_1} \cdots f_s^{i_s}$, where $\alpha_i \in \mathbb{R}_{\geq 0}$ are nonnegative reals (for $i \in \mathbb{N}^s$) such that: there exists i such that $\alpha_i > 0$; and $\alpha_i \neq 0$ for only finitely many i.*

We introduce a notion similar to quadratic module to represent strict cones to prove this lemma.

Definition 3.29 (positive module) An *positive module* generated by $M = (m_\lambda)_{\lambda \in \Lambda} \in \mathbb{R}[\vec{X}]^\Lambda$ (Λ is an index set) is the set

$$\mathcal{PM}(M) \coloneqq \left\{ r + \sum_{\lambda \in \Lambda} r_\lambda m_\lambda \;\middle|\; \begin{array}{l} r \in \mathbb{R}_{\geq 0},\, (r_\lambda)_{\lambda \in \Lambda} \in \mathbb{R}^\Lambda_{\geq 0}, \\ r_\lambda = 0 \text{ except for finitely many } \lambda\text{'s, and} \\ \text{either } r_\lambda > 0 \text{ for at least one } \lambda, \text{ or } r > 0 \end{array} \right\}. \tag{3.25}$$

Then Lemma 3.28 is an immediate consequence of the following lemma.

Lemma 3.30. *Let $f_1, \ldots, f_s \in \mathbb{R}[\vec{X}]$. Then,*

$$\mathcal{SC}(f_1, \ldots, f_s) = \mathcal{PM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbb{N}^s). \tag{3.26}$$

Proof of Lemma 3.28. The left-to-right direction is proved by induction on the construction of $\mathcal{SC}(f_1, \ldots, f_s)$. Let $f \in \mathcal{SC}(f_1, \ldots, f_s)$.

1. *Case:* $f = f_\lambda$ *for some* $\lambda = 1, \ldots, s$.

$$f = f_\lambda = f_1^0 \cdots f_\lambda^1 \cdots f_s^0 \in \mathcal{PM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbb{N}^s). \tag{3.27}$$

2. *Case:* $f = r$ *for some* $r \in \mathbb{R}_{>0}$.

$$f = r \in \mathcal{PM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbb{N}^s). \tag{3.28}$$

3. *Case:* $f = g + h$ *for some* $g, h \in \mathcal{SC}(f_1, \ldots, f_s) \cap \mathcal{PM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbb{N}^s)$. Obvious, because $\mathcal{PM}(f_1^{i_1} \ldots f_s^{i_s} \mid i \in \mathbb{N}^s)$ is closed under addition.
4. *Case:* $f = gh$ *for some* $g, h \in \mathcal{SC}(f_1, \ldots, f_s) \cap \mathcal{PM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbb{N}^s)$. g and h have the form of

$$g = \sum_{i \in \mathbb{N}^s} r_i f_1^{i_1} \cdots f_s^{i_s}, \quad h = \sum_{j \in \mathbb{N}^s} r'_j f_1^{j_1} \cdots f_s^{j_s}, \tag{3.29}$$

where $r_i, r'_j \in \mathbb{R}_{\geq 0}$. Let $q_\lambda = \sum_{i+j=\lambda} r_i r'_j$. Then obviously there are only finitely many λ such that $q_\lambda > 0$. Moreover, there exists λ such that $q_\lambda > 0$. Indeed, for i and j such that $r_i, r'_j > 0$, we have

$$q_{i+j} = \sum_{i'+j'=i+j} r_{i'} r'_{j'} \geq r_i r'_j > 0. \tag{3.30}$$

Therefore,

$$\begin{aligned} gh &= \sum_{i \in \mathbb{N}^s} \sum_{j \in \mathbb{N}^s} r_i r'_j f_1^{i_1+j_1} \cdots f_s^{i_s+j_s} && (3.31)\\ &= \sum_{\lambda \in \mathbb{N}^s} \sum_{i+j=\lambda} r_i r'_j f_1^{i_1+j_1} \cdots f_s^{i_s+j_s} && (3.32)\\ &= \sum_{\lambda \in \mathbb{N}^s} q_\lambda f_1^{\lambda_1} \cdots f_s^{\lambda_s} && (3.33)\\ &\in \mathcal{PM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbb{N}^s). && (3.34) \end{aligned}$$

For the converse, let us consider $f = r + \sum_i r_i f_1^{i_1} \cdots f_s^{i_s} \in \mathcal{PM}(f_1^{i_1} \cdots f_s^{i_s} \mid i \in \mathbb{N}^s)$. For each i such that $r_i > 0$, we have $r_i, f_1^{i_1}, \ldots, f_s^{i_s} \in \mathcal{SC}(f_1, \ldots, f_s)$. Moreover, either $r > 0$ or there exists at least one such i. Therefore we conclude that $f \in \mathcal{SC}(f_1, \ldots, f_s)$, because strict cones are closed under addition and multiplication. ■

To summarize: our analysis of the framework of [39] has led to a new algorithm (Algorithm 2) that allows "sharp" interpolation of barely disjoint SASs. This algorithm is based on strict inequalities ($>$) instead of disequalities ($\neq$); we introduced the corresponding notion of *strict*

cone. The algorithm allows solution by numeric SDP solvers. Moreover we observe that the common variable condition—that seems to be only partially addressed in [39]—can be encoded as SDP constraints.

We conclude by noting that our algorithm (Algorithm 2) generalizes Algorithm 1 from [39]. More specifically, given $\mathrm{SAS}_{\neq}$'s $\mathcal{T}$ and $\mathcal{T}'$ that are disjoint, if Algorithm 1 finds an interpolant, then Algorithm 2 also finds an interpolant after suitable translation of $\mathcal{T}$ and $\mathcal{T}'$ to $\mathrm{SAS}_{<}$'s. See Section 3.3.4 for details.

3.3.4 Relationship of the Two Algorithms

We show that if Algorithm 1 generates an interpolant for two $\mathrm{SAS}_{\neq}$'s, then Algorithm 2 generates an interpolant for two $\mathrm{SAS}_{<}$'s that are equivalent to those two $\mathrm{SAS}_{\neq}$'s. Readers can safely skip over this section.

Proposition 3.31. *Let $\mathcal{T}$ and $\mathcal{T}'$ be the $\mathrm{SAS}_{\neq}$'s in (3.2). Let $\mathcal{U}$ and $\mathcal{U}'$ be the following $\mathrm{SAS}_{<}$'s.*

$$\begin{aligned}
\mathcal{U} &= \left(\begin{array}{l} f_1(\vec{X},\vec{Y}) \geq 0 \ , \ \ldots, \ f_s(\vec{X},\vec{Y}) \geq 0 \ , \quad g_1^2(\vec{X},\vec{Y}) > 0 \ , \ \ldots, \ g_t^2(\vec{X},\vec{Y}) > 0 \ , \\ h_1(\vec{X},\vec{Y}) = 0 \ , \ \ldots, \ h_u(\vec{X},\vec{Y}) = 0 \end{array} \right), \\
\mathcal{U}' &= \left(\begin{array}{l} f'_1(\vec{X},\vec{Z}) \geq 0 \ , \ \ldots, \ f'_{s'}(\vec{X},\vec{Z}) \geq 0 \ , \quad {g'}_1^2(\vec{X},\vec{Z}) > 0 \ , \ \ldots, \ {g'}_{t'}^2(\vec{X},\vec{Z}) > 0 \ , \\ h'_1(\vec{X},\vec{Z}) = 0 \ , \ \ldots, \ h'_{u'}(\vec{X},\vec{Z}) = 0 \end{array} \right).
\end{aligned} \tag{3.35}$$

Obviously we have $[\![\mathcal{T}]\!] = [\![\mathcal{U}]\!]$ and $[\![\mathcal{T}']\!] = [\![\mathcal{U}']\!]$.

Assume there exist polynomials f, f', g, h, h' that satisfy the following conditions (cf. Theorem 3.11).

- $f \in \mathcal{C}(f_1, \ldots, f_s), f' \in \mathcal{C}(f'_1, \ldots, f'_{s'}), g \in \mathcal{M}(g_1, \ldots, g_t, g'_1, \ldots, g'_{t'}), h \in \mathcal{I}(h_1, \ldots, h_u)$, $h' \in \mathcal{I}(h'_1, \ldots, h'_{u'})$, *and*
- $1 + f + f' + g^2 + h + h' = 0$.

Then there exist polynomials $\widetilde{f}, \widetilde{f}', \widetilde{g}, \widetilde{h}, \widetilde{h}'$ that satisfy the following (cf. Theorem 3.24).

- $\widetilde{f} \in \mathcal{C}(f_1, \ldots, f_s, g_1^2, \ldots, g_t^2), \widetilde{f}' \in \mathcal{C}(f'_1, \ldots, f'_s, {g'}_1^2, \ldots, {g'}_t^2), \widetilde{g} \in \mathcal{SC}(g_1^2, \ldots, g_t^2), \widetilde{h} \in \mathcal{I}(h_1, \ldots, h_u), \widetilde{h}' \in \mathcal{I}(h'_1, \ldots, h'_{u'})$ *and*
- $\widetilde{f} + \widetilde{f}' + \widetilde{g} + \widetilde{h} + \widetilde{h}' = 0$.

Proof. Set

$$\widetilde{f} = g^2 + f \ , \quad \widetilde{f}' = f' \ , \quad \widetilde{g} = 1 \ , \quad \widetilde{h} = h \ , \quad \widetilde{h}' = h' \ . \tag{3.36}$$

The equality $\widetilde{f} + \widetilde{f}' + \widetilde{g} + \widetilde{h} + \widetilde{h}' = 0$ is easy. Because g^2 is an SOS and $f \in \mathcal{C}(f_1, \ldots, f_s)$,

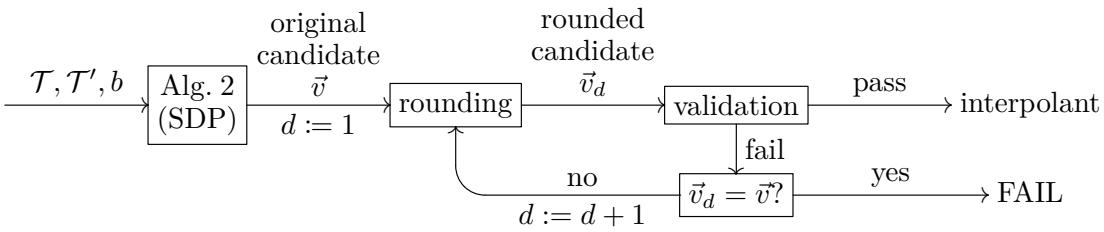


Figure 3.1: The workflow of our tool SSInt

$\widetilde{f} \in \mathcal{C}(f_1, \ldots, f_s, g_1^2, \ldots, g_t^2)$ hold. $\widetilde{f}' \in \mathcal{C}(f_1', \ldots, f_s', {g'}_1^2, \ldots, {g'}_t^2)$ is obvious. By the definition of strict cone, $\widetilde{g} = 1 \in \mathcal{SC}(g_1, \ldots, g_t)$. $\widetilde{h} \in \mathcal{I}(h_1, \ldots, h_u), \widetilde{h}' \in \mathcal{I}(h_1', \ldots, h_{u'}')$ are obvious. ■

Recall that Algorithm 1 generates an interpolant based on Theorem 3.11, and that Algorithm 2 is based on Theorem 3.24. Therefore by Prop. 3.31, if the former succeeds, so does the latter.

3.4 Implementation: Numerical Errors and Rounding

Our implementation, which we named SSInt (*Sharp and Simple Interpolants*), is essentially Algorithm 2, for which we use an SDP solver to solve Line 3. Specifically, we use the SDP solver SDPT3 [127] via YALMIP as backend.

The biggest issue in the course of the implementation is *numerical errors*—they are inevitable due to (numerical) interior point methods underlying most state-of-the-art SDP solvers. On one hand, we often get incorrect interpolants due to numerical errors (Example 3.3). On another hand, in the context of program verification simpler interpolants are often more useful, reflecting simplicity of human insights (see Section 3.1). Numerical solutions, on the contrary, do not very often provide humans with clear-cut understanding.

In our implementation we cope with numerical errors by *rounding* numbers. More specifically we round *ratios* $x_1 : x_2 : \cdots : x_n$ because our goal is to simplify a polynomial inequality $f + g + h > 0$ (imagine the ratio between coefficients). For this purpose we employ an extension of *continued fraction expansion*, a procedure used for the purpose of *Diophantine approximation* [82] (i.e., finding the best rational approximation k_1/k_2 of a given real r). Concretely, our extension takes a ratio $x_1 : \cdots : x_n$ of natural numbers (and a parameter d that we call *depth*), and returns a simplified ratio $y_1 : \cdots : y_n$.

Overall the workflow of our tool SSInt is as in Figure 3.1.

- We first run Algorithm 2. Its output—more precisely the solution of the SDP problem in Line 3—may not yield an interpolant, due to numerical errors. The output is therefore called a *candidate* $\vec{v}$.

Table 3.1: The application of Algorithm 3 to $(871465, 55625, 359255)$

d	$\mathsf{CFE}(x, d)$
1	$(15, 1, 6)$
2	$(31, 2, 13)$
3	$(172, 11, 71)$
4	$(204, 13, 84)$
5	$(11515, 735, 4747)$
6	$(81389, 5195, 33552)$
7	$(174293, 11125, 71851)$
8	$(174293, 11125, 71851)$
x	$(871465, 55625, 359255)$

- We then round the candidate $\vec{v}$ iteratively, starting with the depth $d = 1$ (the coarsest approximation that yields the simplest candidate $\vec{v}_1$) and incrementing it. The bigger the depth d is, the less simple and the closer to the original candidate $\vec{v}$ the rounded candidate $\vec{v}_d$ becomes.
- In each iteration, we check if the candidate $\vec{v}_d$ yields a valid interpolant. This *validation* phase is conducted purely symbolically, ensuring soundness of our tool.
- Our rounding algorithm eventually converges and we have $\vec{v}_d = \vec{v}$ for a sufficiently large d (Lemma 3.32). In case we do not succeed by then we return FAIL.

In other words, we try candidates $\vec{v}_1, \vec{v}_2, \ldots$, from simpler to more complex, until we obtain a symbolically certified interpolant (or fail). This is the *rounding and validation* workflow that we adopted from [58].

Our workflow involves another parameter $c \in \mathbb{N}$ that we call *precision*. It is used as an empirical implementation trick that we apply to the original candidate $\vec{v}$: we round it off to c decimals.

The tool SSInt is implemented in OCaml. When run with SAS$_<$'s $\mathcal{T}$, $\mathcal{T}'$ (and a parameter $b \in \mathbb{N}$ for the maximum degree, see Algorithm 2) as inputs, the tool generates MATLAB code that conducts the workflow in Figure 3.1. The latter relies on the SDP solver SDPT3 [127] via YALMIP as backend.

Algorithm 3 Extended continued fraction expansion CFE

1: **input**: $x = (x_1, \ldots, x_n) \in \mathbb{N}^n$ (at least one of $x_1, \ldots, x_n$ is nonzero), and *depth* $d \in \mathbb{N}_{>0}$
2: Pick p so that x_p is the smallest among the nonzero elements in $x_1, \ldots, x_n$
(say the smallest among such p's)
3: $a \coloneqq (\lfloor x_1/x_p \rfloor, \ldots, \lfloor x_n/x_p \rfloor)$
4: **if** $d = 1$ **then**
5: $y \coloneqq a/\gcd(a)$
6: **return** y
7: **else**
8: $r \coloneqq (x_1 - a_1 x_p, \ldots, x_{p-1} \overset{p-1}{\vee} - a_{p-1} x_p, \overset{p}{\vee} x_p, x_{p+1} \overset{p+1}{\vee} - a_{p+1} x_p, \ldots, x_n - a_n x_p)$
9: $r' \coloneqq \mathsf{CFE}(r, d-1)$ ▷ a recursive call
10: $y \coloneqq (a_1 r'_p + r'_1, \ldots, a_{p-1} r'_p \overset{p-1}{\vee} + r'_{p-1}, \overset{p}{\vee} r'_p, a_{p+1} r'_p \overset{p+1}{\vee} + r'_{p+1}, \ldots, a_n r'_p + r'_n)$
11: **return** $y/\gcd(y)$

3.4.1 Rounding

Continued fraction expansion is a well-known method for rounding a real number to a rational; it is known to satisfy an optimality condition called *Diophantine approximation*. One can think of it as a procedure that simplifies ratios $x_1 : x_2$ of two numbers.

In our tool we use an extension of the procedure to simplify ratios $x_1 : \cdots : x_n$; it is the algorithm CFE in Algorithm 3. An example is in Table 3.1, where $x = (871465, 55625, 359255)$. One sees that the ratio gets more complicated as the depth d becomes bigger. For the depth $d = 7, 8$ the output is equivalent to the input x.

Our algorithm CFE enjoys the following pleasant properties.

Lemma 3.32. *1. (Convergence) The output* $\mathsf{CFE}(x, d)$ *stabilizes for sufficiently large* d*; moreover the limit coincides with the input ratio* x*. That is: for each* x *there exists* M *such that* $\mathsf{CFE}(x, M) = \mathsf{CFE}(x, M+1) = \cdots = x$ *(as ratios).*

2. (Well-definedness) CFE *respects equivalence of ratios. That is, if* $x, x' \in \mathbb{N}^n$ *represent the same ratio, then* $\mathsf{CFE}(x, d) = \mathsf{CFE}(x', d)$ *(as ratios) for each* d*.*

Proof. We first notice that in case where the output of $\mathsf{CFE}(r, d-1)$ in Line 10 equals r, then we can check that the value of y at Line 11 equals x by a straightforward calculation.

We can prove the lemma by induction on $\max\{x_i \mid 1 \leq i \leq n\}$. We check several cases in turn.

1. In the case where there is only one nonzero element in x, set $M = 1$. For simplicity, we consider the case $x = (k, 0, 0)$. Then $p = 1$ and $a = (1, 0, 0)$. Therefore if $d = 1$, then $y = a = x$ (as ratios). If $d > 1$, then $r = (k, 0, 0)$, so by induction on d we obtain $r' = r$, and thus $y = x$ by the remark above.
2. In the case where there are more than one nonzero elements, and all of them are the same, set $M = 2$. For simplicity, we consider the case $x = (k, k, k, 0, 0)$. Then $p = 1$ and $a = (1, 1, 1, 0, 0)$. If $d \geq M = 2$, then else-branch is taken, and $r = (k, 0, 0, 0, 0)$. By the previous case, we have $r' = r$. Therefore $y = x$ by the remark.
3. In other cases, let r as in Line 9, and M be the depth d for which $\mathsf{CFE}(r, d)$ stabilizes (such M exists by the induction hypothesis). Then for any $d \geq M + 1$ we have $r' = r$, and thus $y = x$ by the remark.

The second claim is obvious from the construction of the algorithm. ■

The algorithm CFE takes a positive ratio x as input. In the workflow in Figure 3.1 CFE is applied to ratios with both positive and negative numbers; we deal with such input by first taking absolute values and later adjusting signs.

3.4.2 Validation

Potential unsoundness of verification methods due to numerical errors has been identified as a major challenge (see for example [16, 58, 72, 109, 110, 114]). In our tool we enforce soundness (i.e., that the output is indeed an interpolant) by the validation phase in Figure 3.1.

There the candidate $\vec{v}_d$ in question is fed back to the constraints in (the SDP problem that is solved in) Algorithm 2,[*2] and we check the constraints are satisfied. The check must be symbolic. For equality constraints such a symbolic check is easy. For semidefiniteness constraints, we rely on the following well-known fact: a symmetric real matrix M is positive semidefinite if and only if all the principal minors of M are nonnegative. This characterization allows us to check semidefiniteness using only addition and multiplication. We find no computation in our validation phase to be overly expensive. This is in contrast with QE-based validation methods employed in [38] for example: while symbolic and exact, the CAD algorithm for QE is known to be limited in scalability.

[*2] In Algorithm 2 we introduced the constraint $\sum_{k \in \mathbf{b}^t} \gamma_k \geq 1$ in (3.23) as a relaxation of a natural constraint $\sum_{k \in \mathbf{b}^t} \gamma_k > 0$; see Section 3.3.3. In the validation phase of our implementation we wind back the relaxation $\sum_{k \in \mathbf{b}^t} \gamma_k \geq 1$ to the original constraint with > 0.

3.5 Experiments

We now present some experiment results. In the first part we present some simple geometric examples that call for "sharp" interpolants; in the second we discuss some program verification scenarios. These examples demonstrate our tool's capability of producing simple and sharp interpolants, together with the benefits of such interpolants in program verification techniques.

The experiments were done on Apple MacBook Pro with 2.7 GHz Intel Core i5 CPU and 16 GB memory. As described in Section 3.4, our tool SSInt consists of OCaml code generating MATLAB code; the latter runs the workflow in Figure 3.1. Running the OCaml code finishes in milliseconds; running the resulting MATLAB code takes longer, typically for seconds. The execution time shown here is the average of 10 runs.

Our tool has two parameters: the maximum degree b and the precision c (Section 3.4). In all our examples the common variable condition (in Definition 3.10) is successfully enforced.

3.5.1 Geometric Examples

Table 3.2 summarizes the performance of our tool on interpolation problems. For the input 6, we tried parameters $(b, c) = (1, 1), (1, 2), \ldots, (1, 5)$ and $(2, 5)$ but all failed, leading to FAIL in Figure 3.1. The input 9 contains disjunctions, which is not allowed in $\mathrm{SAS}_<$'s. It is dealt with using the technique described in [39, Section 3.1]: an interpolant of $\mathcal{T}$ and $\mathcal{T}'$ is given by $\bigvee_i \bigwedge_j \mathcal{S}_{ij}$, where $\mathcal{S}_{ij}$ is an interpolant of each pair of disjuncts $\mathcal{T}_i$ and $\mathcal{T}'_j$ of $\mathcal{T}$ and $\mathcal{T}'$, respectively.

Listing 3.1: Code 1.3 of [39]

```
real x,y;
real xa = 0;
real ya = 0;
while(nondet()){
 x = xa + 2*ya;
 y = -2*xa + ya;
 x++;
 if(nondet()){
  y = y + x;
 }else{
  y = y - x;
```

Table 3.2: Experiment results. $\mathcal{T}$ and $\mathcal{T}'$ are inputs, and $\mathcal{S}$ is our output (see Figure 3.2 too). The "time" column shows the execution time (in seconds) of the generated MATLAB code, b and c show the successful choice of parameters, and d is the depth for which the workflow in Figure 3.1 terminated.

	$\mathcal{T}$	$\mathcal{T}'$	$\mathcal{S}$	time [s]	b	c	d
1	$y > x, x > -y$	$0 \geq y$	$4y > 0$	2.19	0	5	1
2	$y \leq 0$	$y > x^2$	$-2y \geq 0$	5.68	2	3	1
3	$y > x, x > -y$	$y \leq x, x \leq -y$	$4y > 0$	2.67	0	5	1
4	$y > x, x > -y$	$y \leq -x^2$	$8y + 4x^2 > 0$	5.09	2	1	1
5	$y \leq -1$	$x^2 + y^2 < 1$	$34y^2 - 68y - 102 \geq 0$	7.58	2	5	3
6	$x^2 + (y-1)^2 \leq 1$	$x^2 + (y-2)^2 > 4$	FAIL	14.0	2	5	8
7	$x^2 + (y+1)^2 \leq 1$	$x^2 + (y-1)^2 < 1$	$18x^2y - 14x^2y^2 - 144y$ $+28y^2 - 7x^4 + 18y^3 - 7y^4 \geq 0$	6.45	2	2	2
8	$x \geq z^2$	$x < -y^2$	$2x \geq 0$	7.67	2	3	1
9	$(y \geq (x-1)^2) \vee$ $(y > (x+1)^2)$	$(y < -(x-1)^2) \vee$ $(y \leq -(x+1)^2)$	$((586x + 293y + 119 > 0) \wedge (333y \geq 0)) \vee$ $((333y > 0) \wedge (374y - 748x - 117 \geq 0))$	43.7	2	3	3

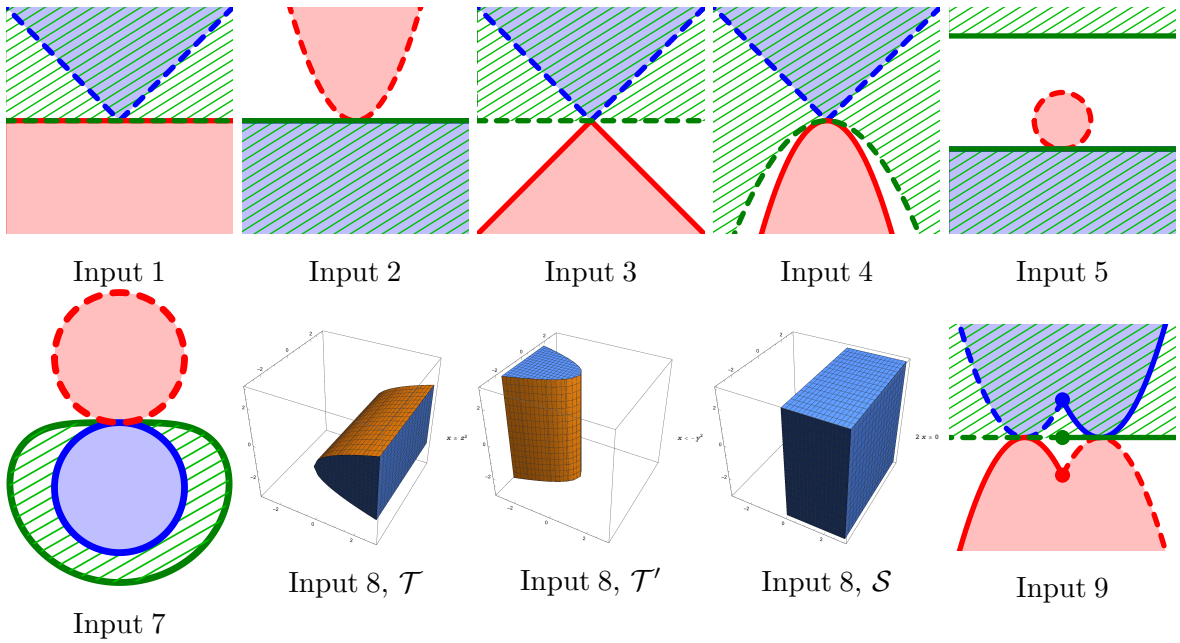


Figure 3.2: Interpolants from Table 3.2. The blue, orange and green areas are for $\mathcal{T}$, $\mathcal{T}'$, $\mathcal{S}$, respectively.

```
 }
 xa = x - 2*y;
 ya = 2*x + y;
}
assert(xa + 2*ya >= 0);
```

Listing 3.2: Constant Acceleration

```
real x,v;
(x, v) = (0, 0);
while(nondet()){
 (x, v) = (x+2*v, v+2);
}
 assert(x >= 0);
```

3.5.2 Program Verification Example I: Infeasibility Checking

Consider the code in Listing 3.1; this is from [39, Section 7]. We shall solve Subproblem 1 in [39, Section 7]: if the property $(xa) + 2(ya) \geq 0$ holds at Line 5, then it holds too after the execution along $5 \to 6 \to 7 \to 8 \to 9 \to 13 \to 14$. The execution is expressed as the SAS $\mathcal{T}$:

$$\mathcal{T} = \left(\begin{array}{l} (xa) + 2(ya) \geq 0, \;\; x = (xa) + 2(ya), \\ y = -2(xa) + (ya), \;\; x_1 = x + 1, \\ y_1 = x_1 + y, \;\; (xa_1) = x_1 - 2y_1, \\ (ya_1) = 2x_1 + y_1 \end{array} \right).$$

Then our goal is to show that the negation $\mathcal{T}' = ((xa_1) + 2(ya_1) < 0)$ of the desired property is disjoint from $\mathcal{T}$.

Our tool yields $\mathcal{S} = \big(8 - 14(ya_1) - 7(xa_1) \geq 0\big)$ as an interpolant of these $\mathcal{T}$ and $\mathcal{T}'$ (in 14.1 seconds, with parameters $b = 0, c = 3$ and depth $d = 11$). The interpolant witnesses disjointness. Our interpolant is far simpler than the interpolant given in [39].[*3]

Here the simplicity of our interpolant brings *robustness* as its benefit. Consider the other path $5 \to \cdots \to 8 \to 11 \to 13 \to 14$ of execution from Line 5 to 14, and let $\mathcal{T}_0$ be the SAS that expresses the execution similarly to $\mathcal{T}$ in the previous example. It turns out that

[*3] An interpolant $716.77 + 1326.74(ya) + 1.33(ya)^2 + 433.90(ya)^3 + 668.16(xa) - 155.86(xa)(ya) + 317.29(xa)(ya)^2 + 222.00(xa)^2 + 592.39(xa)^2(ya) + 271.11(xa)^3 > 0$ is given in [39]. We note that, to show disjointness of $\mathcal{T}$ and $\mathcal{T}'$, an interpolant of any splitting of $\mathcal{T} \cup \mathcal{T}'$ would suffice. It is not specified in [39] which splitting they used.

our interpolant $\mathcal{S}$ in the above is also an interpolant of $\mathcal{T}_0$ and $\mathcal{T}'$. Thus our algorithm has managed, aiming at simpler interpolants, to automatically discover $-14(ya) - 7(xa)$ (that is, $(xa) + 2(ya)$) as a value that is significant regardless of the choice made in Line 8.

3.5.3 Program Verification Example II: CEGAR

This is the example we discussed in Example 3.4. Here we provide further details, aiming at readers familiar with CEGAR.

One of the most important applications of interpolation in verification is in *counterexample-guided abstraction refinement (CEGAR)* [32]. There an interpolant $\mathcal{S}$ is used as (a candidate for) the "essential reason" to distinguish positive examples $\mathcal{T}$ from negative counterexamples $\mathcal{T}'$.

As an example let us verify Listing 3.1 by CEGAR. Starting from the empty set of abstraction predicates, CEGAR would find the path $p_1 := (1 \to 2 \to 3 \to 4 \to 16)$ as a counterexample.[*4] This counterexample path turns out to be spurious: let $\mathcal{T} := (xa = 0, ya = 0)$ express the path and $\mathcal{T}' := ((xa) + 2(ya) < 0)$ express the negation of the assertion; our tool SSInt yields $189346(xa) + 378692(ya) \geq 0$ (i.e. $(xa) + 2(ya) \geq 0$) as an interpolant, proving their disjointness. For the interpolation the tool SSInt took 4.32 seconds; we used the parameters $b = 0$ and $c = 5$.

Consequently we add $(xa) + 2(ya) \geq 0$ as a new abstraction predicate and run the CEGAR loop again. This second run succeeds, since $(xa) + 2(ya) \geq 0$ turns out to be a suitable invariant for the loop in Line 4. We conclude safety of Listing 3.1.

We tried to do the same with the tool `aiSat` [37, 39] instead of our SSInt. It does not succeed in interpolating $\mathcal{T} = (xa = 0, ya = 0)$ and $\mathcal{T}' = ((xa) + 2(ya) < 0)$, since sharpness is required here (Prop. 3.18). As a workaround we tried strengthening $\mathcal{T}' = ((xa) + 2(ya) < 0)$ into $\mathcal{T}'_0 = ((xa) + 2(ya) \leq -10^{-7})$; `aiSat` then succeeded and yielded an interpolant $\mathcal{S} = (137.3430 + 5493721088(ya) + 2746860544(xa) > 0)$. This predicate, however, cannot exclude the spurious path p_1 because $\mathcal{S}$ and the negation $(xa)+2(ya) < 0$ of the assertion are satisfiable with $xa = 0$ and $ya = -1.25 \times 10^{-8}$.

[*4] Here we use a *path-based* CEGAR workflow that uses an execution path as a counterexample. Since we do not have any abstraction predicates, xa and ya can be any integers; in this case the assertion in Line 16 can potentially fail.

3.5.4 Program Verification Example III: CEGAR

Here is another CEGAR example. Consider the code in Listing 3.2 that models movement with constant acceleration. We initially have the empty set of abstraction predicates. After the first run of the CEGAR loop we would obtain a counterexample path $p_1 := (1 \to 2 \to 3 \to 6)$; note that, since there are no predicates yet, x can be anything and thus the assertion may fail.

We let $\mathcal{T}_1 := (x = 0, v = 0)$ express the counterexample p_1 and $\mathcal{T}_1' := (x < 0)$ express the negation of the assertion. For these $\mathcal{T}_1, \mathcal{T}_1'$ our tool SSInt synthesizes $\mathcal{S}_1 := (2x \geq 0)$ as their interpolant (in 1.92 seconds, with $b = 0$, $c = 5$, and $d = 1$).

Thus we add $2x \geq 0$ as an abstraction predicate and run the CEGAR loop again. We would then find the path $p_2 := (1 \to 2 \to 3 \to 4 \to 5 \to 6)$ as a counterexample—note that the previous counterexample p_1 is successfully excluded by the new predicate $2x \geq 0$. Much like before, we let $\mathcal{T}_2 := (v_1 = 0)$ express an initial segment of the path p_2 and let $\mathcal{T}_2' := (x_1 = 0, v_2 = v_1 + 2, x_2 = x_1 + 2v_1, x_2 < 0)$ express the rest of the path p_2 (together with the negation of the assertion), and we shall look for their interpolant $\mathcal{S}_2$ as the witness of infeasibility of the path p_2. SSInt succeeds, yielding $\mathcal{S}_2 := (8v_1 \geq 0)$ in 2.87 seconds with $b = 0, c = 5, d = 1$.

In the third run of the CEGAR loop we use both $2x \geq 0$ and $8v \geq 0$ (from $\mathcal{S}_1, \mathcal{S}_2$) as abstraction predicates. The proof then succeeds and we conclude safety of Listing 3.2.

We did not succeed in doing the same with `aiSat`. In the first CEGAR loop an interpolant of $\mathcal{T}_1$ and $\mathcal{T}_1'$ cannot be computed because it has to be sharp. As we did in the previous example we could strengthen $\mathcal{T}_1'$ to $\mathcal{T}_1'' := (x \leq 10^{-7})$ and use an interpolant of $\mathcal{T}_1$ and $\mathcal{T}_1''$ instead for the next iteration. `aiSat` generated an interpolant $3790.1050 + 75802091520.0000x > 0$ of $\mathcal{T}_1$ and $\mathcal{T}_1''$; however this fails to exclude the spurious counterexample path p_1.

Overall this example demonstrates that sharpness of interpolants can be a decisive issue in their application in program verification.

3.6 Conclusions

We proposed a new method to find an polynomial interpolant for a pair of semialgebraic systems which can be "barely disjoint". As the existing method by Dai et al. does not generate any interpolants in this case and our method works when the existing method works, our method is a proper extension of Dai et al.'s method. For our method, we proposed *strict cone*, which is an algebraic structure in $\mathbb{R}[\vec{X}]$, and designed an numerical optimization problem with it.

To find an exact and simple interpolant, we proposed a technique to round the result of the numerical optimization with an extension of the fractional expansion.

3.7 Future Work

The workflow of the rounding-validation loop [58] is simple but potentially effective: in combination with our rounding algorithm based on continued fractions, we speculate that the workflow can offer a general methodology for coping with numerical errors in verification and in symbolic reasoning. Certainly our current implementation is not the best of the workflow: for example, the validation phase of Section 3.4 could be further improved by techniques from interval arithmetic, as in [115].

The collaboration between numerical and symbolic computation in general (as in [4]) interests us, too. For example in our workflow (Figure 3.1, pp. 61) there is a disconnection between the SDP phase and later: passing additional information (such as gradients) from the SDP phase can make the rounding-validation loop more effective.

Our current examples are rather simple and small. While they serve as a feasibility study of the proposed interpolation method, practical applicability of the method in the context of program verification is yet to be confirmed. We plan to conduct more extensive case studies, using common program verification benchmarks such as in [121], making comparison with other methods, and further refining our method in its course.

Chapter 4

Mind the Gap: Bit-vector Interpolation Recast over Linear Integer Arithmetic

4.1 Introduction

4.1.1 Interpolant

For a pair of logical formula A and B such that A and B are inconsistent, an interpolant I of A and B is a formula such that (1) A implies I, (2) B and I are inconsistent, and (3) the variables in I are covered with the intersection of the variables of A and the variables of B. From a set-theoretic viewpoint, the conditions (1) and (2) means that

$$\{\, p \mid A(p) \text{ holds} \,\} \subseteq \{\, p \mid I(p) \text{ holds} \,\} \subseteq \{\, p \mid B(p) \text{ holds} \,\}^{c} . \tag{4.1}$$

As the condition (3) restricts the expressivity of I, an interpolant of A and B means "a simple formula separating the region of A and B". Note that many different interpolants can exist for a pair $\langle A, B \rangle$.

For example, in propositional logic, letting $A = (P_1 \wedge P_2 \wedge P_4)$ and $B = \neg(P_1 \vee P_2 \vee P_3)$, formulae $I_1 = P_3$ and $I_2 = (P_1 \vee P_3)$ are interpolants of A and B. By the Karnaugh map shown in Table 4.1, we can easily check that I_1 and I_2 are interpolants. Note that A does not satisfy (3), so it is not an interpolant of A and B even though it satisfies (1) and (2).

We can think of interpolants in the theory of linear inequalites. For example, letting $A = (x = y + 1) \wedge (y = 0)$ and $B = (x = z + 2) \wedge (1 \leq z)$, formulae $I_1 = (x = 1)$, $I_2 = (x \leq 1)$ and $I_3 = (x < 3)$ are interpolants of A and B.

Table 4.1: The Karnaugh map of A, B, I_1, I_2

	P_3, P_4	$P_3, \overline{P_4}$	$\overline{P_3}, \overline{P_4}$	$\overline{P_3}, P_4$
P_1, P_2	A, I_1, I_2	I_1, I_2	I_2	I_2
$P_1, \overline{P_2}$	A, I_1, I_2	I_1, I_2	I_2	I_2
$\overline{P_1}, \overline{P_2}$	I_1, I_2	I_1, I_2	B	B
$\overline{P_1}, P_2$	I_1, I_2	I_1, I_2		

Interpolant has been used as a technique for program verification [92]. Specifically, interpolant computation is combined with the techniques of constructing graphs between abstract program states to discover invariants [51] and lazy abstractions [60], and interpolant computation is performed to discover propositions in those techniques that are useful for their verification. BLAST [17] and IMPACT [89] have been produced as successful tools using such techniques.

For example, in the case of IMPACT [17], it constructs a program invariant by preparing a graph whose nodes are pairs of program counters and overapproximations of the conditions to reach the nodes, and then refine those conditions. This technique uses an SMT solver to determine whether the path from the initial state to an error state on the graph is possible or impossible due to an incorrect overapproximation. If the latter occurs, i.e., the overapproximation is incorrect, then the program's sequence of instructions along the path does not actually occur, and the sequence of propositions converted from the instruction sequence becomes unsatisfiable. By calculating interpolants from the sequence of propositions and strengthening the conditions of the nodes on the path by the interpolants, the abstraction of the program becomes elaborate, and the verification of the program proceeds.

The condition that the variables of an interpolant have to be covered with the intersection of the variables of the original formulae makes the interpolant simple, and it is considered that the simplicity contributes to extracting the essence of the target program.

4.1.2 Context

The interpolant computation has been investigated for various theories [29, 30, 67, 73, 78, 88, 116]. Among them, [52] and [8] worked on the bit-vector theory [8]. Backeman et al. developed a richer theory based on Presburger arithmetic, and proposed a method to calculate interpolants over the theory. (Hence, the work is not actually an interpolation over the bit-vector theory.)

Griggio proposed a method to calculate interpolants over the bit-vector theory by the fol-

lowing steps: (1) convert the target pair of formulae into the pair of formulae over linear integer arithmetic (LIA) preserving the semantics, (2) calculate an interpolant of the pair of the converted formulae by existing LIA interpolation engines [53], and (3) translate the interpolant back to the bit-vector theory [52]. The advantage of the method compared to bit-blasting is the compactness and the "beautifulness" (its benefit is discussed in [2]) of resulting interpolants derived from respecting the syntax of the target formulae. A challenge reported in his paper is that translating back in (3) does not always succeed. When it fails, the method uses bit-blasting as a backup. The failure is that translating back is done by reinterpreting the interpolant in the bit-vector theory as a formula in LIA without changing the syntax. The two semantics can differ mainly because of the overflow: integers in LIA are unbounded, but integers in the bit-vector are bounded, and the wraparound can happen.

4.1.3 Contribution

We propose a technique for transforming LIA formulae into BV formulae while preserving semantics so that the step of Griggio's approach of transforming LIA interpolants into BV formulae always succeeds.

First, we define the problem precisely, observe that the cause of the overflow problem consists mainly of formulae involving inequalities, and we see that the transformation of inequalities of a particular form is essential. Next, we see that there is an obvious way to convert an LIA formula into a BV (bit-vector) formula while preserving semantics, but the obvious approach destroys the structure of the original LIA formula and further increases the size of the expression drastically. Next, we discuss the first element of our approach: *boxing*. We see that, for inequalities in the LIA satisfying a special condition, we can create equivalent LIA inequalities such that the naïve transformation to a BV formula preserves its semantics by a small modification. This result shows that if an LIA inequality satisfies the special condition, then the LIA inequality can be converted into the corresponding BV formula without any problem concerning the preservation of its semantics. Next, we discuss the second element of our approach: *gapping* [*1]. This is a technique to transform arbitrary LIA inequalities into combinations of LIA inequalities satisfying the special condition of the boxing. Therefore, by using "boxing & gapping", we can convert arbitrary LIA inequalities into BV formulas while preserving their semantics. Finally, we discuss how to deal with the remaining cause of conversion failure, namely, expressions with mods.

To check the effectiveness of the above theoretical approach, we implemented an IMPACT-

[*1] The title of the chapter alludes to both this geometric technique, the conceptual gap in previous work, and collaboration which entailed traveling through London.

based program verifier using Griggio's interpolation technique. In the first experiment, we show that our "boxing & gapping" process does not take much time by comparing it with program verification in LIA (in this case, we simplify the program interpretation). In the second experiment, we see that the verification in bit-vector theory is faster with "boxing & gapping" than with the obvious conversion described above.

In summary, our contributions can be summarized as follows:

- We propose a method to calculate a compact interpolant of BV formulae by using a method to compute LIA interpolants.
- We propose and mathematically justify the conversion of a BV formula into an LIA formula without increasing its size too much while preserving its semantics.
- We implement an IMPACT-based [89] program verifier using our interpolant computation technique and show the effectiveness of our technique with experiments.

4.1.4 Use Case (Motivation)

We explain the motivation for using our technique, which realizes interpolation within BV theory, instead of embedding formulae in another theory supporting interpolation and compute interpolants in the theory.

First, Griggio's technique is an ensemble method that layers multiple BV interpolation techniques, and the propositions used in his program verification are in the BV theory. Our method can generate BV interpolants rather than computing interpolants on a different theory as Backeman does. Thus, we can naturally incorporate our method as one of the layers of Griggio's technique.

Secondly, the interpolant calculation over the LIA is still being developed and the possibility of bugs in SMT solvers cannot be ruled out, so we cannot be sure that the interpolant in LIA is correct[*2]. If we generate a BV interpolant by our method, we can double-check the validity of the interpolant in both the LIA and the BV theory.

4.1.5 Related Work

The problem of reasoning about machine arithmetic and wrapping arises not only in model checking, but abstract interpretation too, where solvers are augmented with support for re-

[*2] The only SMT solver that currently supports interpolant computation over LIA is MathSAT5 [28], but its source code is not publicly available. We have reported one bug that causes a crash during our experiments with MathSAT5.

laxing abstraction rather than interpolation.

Despite the long-standing work [14, 20, 96] in deciding BV theories, there has been a lot of work on BV interpolation. Although not focussing on BV interpolation, early work on deriving work-level interpolants [78] uses bit-vectors to interpolate equality logic. This logic supports equations of the form $x = y$ and $x = c$ where x and y are variables and c is drawn from a finite set of symbols C. Bit-vectors with width $\lceil \log_2(|C|) \rceil$ are used to bit-blast equations [111] so that formulae are encoded entirely propositionally. Then a propositional resolution proof of the inconsistency of two formulae is lifted to the work-level.

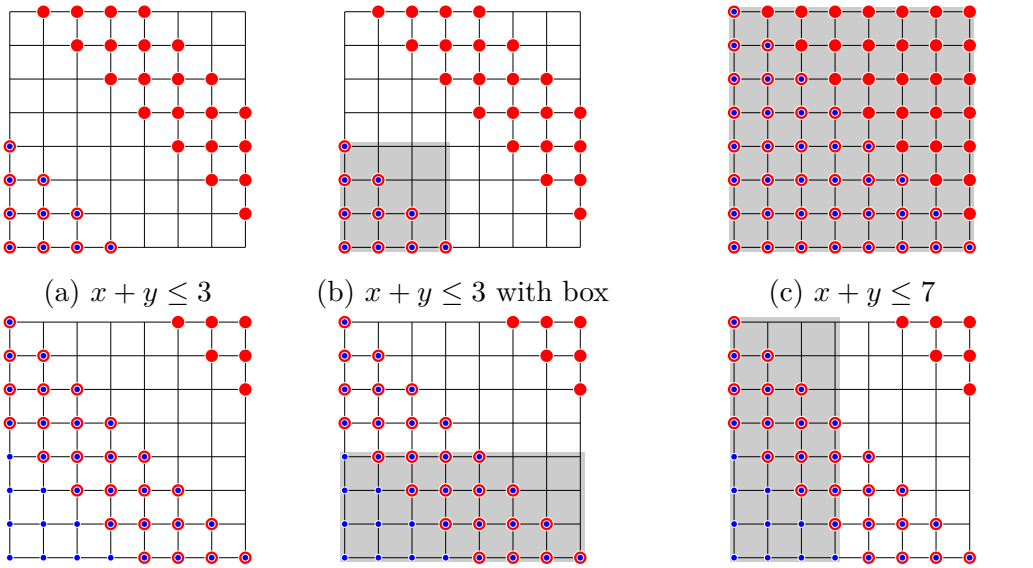
Seminal work by Griggio [52] advocated encoding BV formulae in theories of increasing complexity. The pair of BV formulae are encoded in a theory whose interpolation engine is used to find an interpolant in that theory. The interpolant is then reinterpreted as a BV formula and tested to see if it is still an interpolant for the pair of BV formulae. The approach resorts to bit-blasting if no simpler theory can find an interpolant, at the cost of losing world-level information. By way of contrast, Backeman et al. [8] proposed a calculus over a core language, which supports interpolation and is rich enough to describe BV formulae, even making use of Groebner bases to express polynomial equality relationships. Since interpolation is performed within their core language, they do not aim to derive a BV interpolant, and therefore their work is orthogonal to ours. Yet, if Backeman's procedure returns an interpolant in their core language and if this interpolant could be interpreted as an LIA formula, which seems likely in many cases, then our work would convert the LIA formula back to the BV theory.

Further afield, polynomial algorithms for interpolation have been developed for systems of linear congruence equations [67, section 4], conjunctions of linear Diophantine equations and disequations [67, section 6], and systems of mixed integer linear equations [67, section 7]. This comprehensive study stops short of using LIA to interpolate BV formulae, mentioning the problem as future work.

Abstract domains have been proposed for tracking linear modulo relationships where the module is a power of 2 [47, 76, 97]. These domains, which are essentially specialist solvers, express more than linear equalities [74], while enabling the domain operations to be realized using the machine arithmetic. Surprisingly, systems of linear inequalities can be reinterpreted to the model machine arithmetic by just changing the concretization function [123] and the handling of guards [123].

4.1.6 Organization of the Chapter

In Section 4.2, we give the intuition of boxing and gapping by simple examples. In Section 4.3, we give a formal discussion of our approach. In Section 4.4, we give the experimental



(a) $x + y \leq 3$ (b) $x + y \leq 3$ with box (c) $x + y \leq 7$

(d) $x + y - 4 \leq 3$ (e) $x + y - 4 \leq 3$ with box 1 (f) $x + y - 4 \leq 3$ with box 2

Figure 4.1: Gapping and boxing for $x + y \leq 3$ and $x + y \leq 7$

result. Section 4.5 concludes, and Section 4.6 presents future work.

4.2 Boxing and Gapping in Pictures

We consider the problem of finding a BV equation ℓ such that for a given linear inequality f, $[\![f]\!]_{\mathsf{LIA}} = [\![\ell]\!]_{\mathsf{BV}}$, where $[\![f]\!]_{\mathsf{LIA}}$ is the set of points satisfying f in the LIA and $[\![\ell]\!]_{\mathsf{BV}}$ is the set of points satisfying ℓ in the BV theory. Roughly speaking, we can usually interpret formulae in the usual way in the LIA, and we have to take into account the overflow to interpret formulae in the BV theory (the precise definition is given in the next section). Our task is to find efficiently a formula ℓ of compact size, where the size of a formula is measured by the number of its atomic propositions.

We are explaining the idea of our technique *boxing* and *gapping* with motivating examples. We assume here that unsigned integers are in $[0, \ldots, m-1]$. First, letting $m = 8$ and $f = (x + y \leq 3)$, we are finding ℓ such that $[\![f]\!]_{\mathsf{LIA}} = [\![\ell]\!]_{\mathsf{BV}} \subseteq [0, \ldots, 7 (= m-1)]^2$. The points of $[\![f]\!]_{\mathsf{LIA}}$ are shown as blue points in Figure 4.1(a). Note that $[\![f]\!]_{\mathsf{LIA}} \neq [\![f]\!]_{\mathsf{BV}}$, so we cannot set $l = f$, as $\langle x, y \rangle = \langle 3, 5 \rangle \in [\![f]\!]_{\mathsf{BV}} \setminus [\![f]\!]_{\mathsf{LIA}}$ ($[\![f]\!]_{\mathsf{BV}}$ is shown as red points in Figure 4.1(a)). The reason of the belonging $\langle 3, 5 \rangle \in [\![f]\!]_{\mathsf{BV}}$ is that the overflow in the BV theory as

$$x + y = (3 + 5) \bmod m = 0 \leq 3. \tag{4.2}$$

Hence, finding ℓ is not an obvious problem.

4.2.1 Enumeration

A straightforward way is to enumerate points of $[\![f]\!]_{\mathsf{LIA}}$ and represent them as the formula ℓ. If $f = (x + y \leq 3)$, ℓ can be chosen as

$$\begin{aligned}\ell =&[(x = 0 \wedge y = 0) \vee (x = 0 \wedge y = 1) \vee (x = 0 \wedge y = 2) \vee (x = 0 \wedge y = 3) && (4.3)\\ &\vee (x = 1 \wedge y = 0) \vee (x = 1 \wedge y = 1) \vee (x = 1 \wedge y = 2) && (4.4)\\ &\vee (x = 2 \wedge y = 0) \vee (x = 2 \wedge y = 1) && (4.5)\\ &\vee (x = 3 \vee y = 0)]. && (4.6)\end{aligned}$$

In this way, there is no concern of overflow because there are no numerical operations, and obviously $[\![f]\!]_{\mathsf{LIA}} = [\![\ell]\!]_{\mathsf{BV}}$ holds. The order of the size of ℓ is approximately m^d.

A little better way is to make "pillars" by getting the interval of y for each x, and enumerate the pillars. If $f = (x + y \leq 3)$, ℓ is

$$\ell = [(x = 0 \wedge y \leq 3) \vee (x = 1 \wedge y \leq 2) \vee (x = 2 \wedge y \leq 1) \vee (x = 3 \wedge y \leq 0)]. \tag{4.7}$$

This method can be applied recursively even if the number of variables in f is larger than two. Although this method is more efficient than the previous enumerating method, the order of the size of ℓ is still large.

4.2.2 Boxing

Observing Figure 4.1(a), we see that the points of $[\![f]\!]_{\mathsf{BV}} \setminus [\![f]\!]_{\mathsf{LIA}}$, which are shown as purely red points in the figure, appears as a band in the upper right corner. Thus, we add a "box" predicate $x \leq 3 \wedge y \leq 3$ into f, and make ℓ be

$$\ell = (x + y \leq 3 \wedge (x \leq 3 \wedge y \leq 3)), \tag{4.8}$$

then $[\![f]\!]_{\mathsf{LIA}} = [\![\ell]\!]_{\mathsf{BV}}$ holds (see Figure 4.1(b)). This works because no numerical operations appear in the box $x \leq 3 \wedge y \leq 3$ and there is no concern of overflow. This technique reduces the growth in the size of ℓ. For example, if we consider the case $m = 1024$ and $f = (x + y \leq 511)$, the corresponding ℓ is

$$\ell = (x + y \leq 511 \wedge (x \leq 511 \wedge y \leq 511)). \tag{4.9}$$

This technique is called *boxing* and formalized in the next section.

This boxing described above may not always be applicable. For example, letting $f = (x+y \leq 7)$, there is no gap between $[\![f]\!]_{\mathsf{LIA}} \cap [\![f]\!]_{\mathsf{BV}}$ (red and blue points) and $[\![f]\!]_{\mathsf{BV}} \setminus [\![f]\!]_{\mathsf{LIA}}$ (purely red points) in Figure 4.1(c), and we cannot separate these two sets by a box.

4.2.3 Gapping

Our *gapping* is a technique to transform a linear inequality f into a formula f' such that $[\![f]\!]_{\mathsf{LIA}} = [\![f']\!]_{\mathsf{LIA}}$ and the boxing above is applicable. This solves the above problem. We are applying the gapping to the case $f = (x + y \leq 7)$ and $m = 8$ as an example. As $[0, \ldots, 7] = [0, \ldots, 3] \cup [4, \ldots, 7]$,

$$[\![x + y \leq 7]\!]_{\mathsf{LIA}} = [\![x + y \leq 3]\!]_{\mathsf{LIA}} \cup [\![4 \leq x + y \leq 7]\!]_{\mathsf{LIA}} = [\![x + y \leq 3]\!]_{\mathsf{LIA}} \cup [\![x + y - 4 \leq 3]\!]_{\mathsf{LIA}}. \tag{4.10}$$

The first set on the righthand side is the same as the example in the previous section so that our boxing is applicable. The second set on the righthand side looks like Figure 4.1(d). By adding a pair of boxes $x \leq 3 \vee y \leq 3$ into f and making ℓ be

$$\ell = [((x + y \leq 3) \wedge (x \leq 3 \wedge y \leq 3)) \vee ((x + y - 4 \leq 3) \wedge (x \leq 3 \vee y \leq 3))], \tag{4.11}$$

we can find ℓ such that $[\![f]\!]_{\mathsf{LIA}} = [\![\ell]\!]_{\mathsf{BV}}$ (see Figure 4.1(e, f)).

This gapping is also formalized in the next section.

4.3 Formal Correctness of Boxing and Gapping

We use $x_1, \ldots, x_d$ as variables in the linear inequalities for some $d > 1$. We consider bit-vectors of fixed-width $w > 1$ and interpret LIA and BV formulae over the product space $\mathbb{M}^d$ where $\mathbb{M} = \{\, 0, 1, 2, \ldots, m - 1 \,\}$ and $m = 2^w$ as follows:

Definition 4.1. Let $\vec{c}, \vec{c'} \in \mathbb{Z}^d$ and $b, b' \in \mathbb{Z}$. If $\ell \equiv (\sum_{i=1}^{d} c_i x_i) + b \leq (\sum_{i=1}^{d} c'_i x_i) + b'$ then

$$[\![\ell]\!]_{\mathsf{LIA}} = \{\, \vec{x} \in \mathbb{M}^d \,\big|\, \textstyle\sum_{i=1}^{d} c_i x_i + b \leq \sum_{i=1}^{d} c'_i x_i + b' \,\}$$

$$[\![\ell]\!]_{\mathsf{BV}} = \{\, \vec{x} \in \mathbb{M}^d \,\big|\, (\textstyle\sum_{i=1}^{d} c_i x_i + b) \bmod m \leq (\sum_{i=1}^{d} c'_i x_i + b') \bmod m \,\}$$

Furthermore, the LIA semantics can be lifted from inequalities to LIA formulae by: $[\![f_1 \vee f_2]\!]_{\mathsf{LIA}} = [\![f_1]\!]_{\mathsf{LIA}} \cup [\![f_2]\!]_{\mathsf{LIA}}$, $[\![f_1 \wedge f_2]\!]_{\mathsf{LIA}} = [\![f_1]\!]_{\mathsf{LIA}} \cap [\![f_2]\!]_{\mathsf{LIA}}$ and $[\![\neg f]\!]_{\mathsf{LIA}} = \mathbb{M}^d \setminus [\![f]\!]_{\mathsf{LIA}}$. Likewise for BV formulae.

In the sequel, $\mathbb{N}$ denotes the set of (strictly) positive integers, $\mathbb{R}$ the set of real numbers, and $\mathbb{R}_{\geq 0}$ the set of non-negative real numbers. We extend the floor and ceiling function to sequences in $\mathbb{R}^d$ in a component-wise manner: $\lfloor \vec{x} \rfloor_i = \lfloor x_i \rfloor$ and $\lceil \vec{x} \rceil_i = \lceil x_i \rceil$. If $\vec{x} \in \mathbb{R}^d$ then $|\vec{x}| = d$. The partial order $\leq$ on $\mathbb{R}^d$ is defined by $\vec{x} \leq \vec{y}$ if and only if $x_i \leq y_i$ for all $i = 1, \ldots, d$.

4.3.1 Boxing

The following lemma tells us the boxing in a very basic case, which is the case when the coefficients of the left-hand side of the linear inequality f are all ones and the right-hand side is a constant:

Lemma 4.2. Let $d > 1$ and $L \in \mathbb{N}$. Then:

$$\begin{aligned} & \{\, \vec{x} \in \mathbb{R}^d_{\geq 0} \,\Big|\, \textstyle\sum_{i=1}^d x_i \leq L \cdot (m/2) - 1 \,\} \\ \subseteq\ & \textstyle\bigcup_{\vec{p} \in I_d((d-1)(L+1))} \bigcap_{i=1}^d \{\, \vec{x} \in \mathbb{R}^d_{\geq 0} \mid x_i < \frac{p_i \cdot (m/2)}{d-1} \,\} \\ \subseteq\ & \{\, \vec{x} \in \mathbb{R}^d_{\geq 0} \,\Big|\, \textstyle\sum_{i=1}^d x_i < (L+1) \cdot (m/2) \,\} \end{aligned}$$

where $I_d(n) = \{\, (i_1, \ldots, i_d) \in \mathbb{N}^d \mid i_1 + \cdots + i_d = n \,\}$.

Proof. To prove the first inclusion, let $\vec{x} \in \mathbb{R}^d_{\geq 0}$ satisfy $\sum_{i=1}^d x_i \leq L \cdot (m/2) - 1$. Let $a_i = \lfloor 2(d-1)x_i/m + 1 \rfloor$ for each $i = 1, \ldots, d$. Since the elements of $\vec{x}$ are nonnegative, the elements of $\vec{a}$ are strictly positive.

$$\begin{aligned} \sum_{i=1}^d a_i &\leq \sum_{i=1}^d \left(\frac{(d-1)x_i}{m/2} + 1 \right) && (4.12) \\ &= \frac{d-1}{m/2} \left(\sum_{i=1}^d x_i \right) + d && (4.13) \\ &\leq \frac{d-1}{m/2} (L(m/2) - 1) + d && (4.14) \\ &= (d-1)L - \frac{d-1}{m/2} + d && (4.15) \\ &= (L+1)(d-1) + \left(1 - \frac{d-1}{m/2} \right). && (4.16) \end{aligned}$$

By assumption $1 < d$, $1 - (d-1)/(m/2) < 1$ holds. Since $\sum_{i=1}^d a_i$ is integral, $\sum_{i=1}^d a_i \leq (L+1)(d-1)$ holds. We define $\vec{p}$ by

$$p_i = \begin{cases} (d-1)(L+1) - \sum_{j=2}^d a_j & i = 1 \\ a_i & i > 1 \end{cases} \tag{4.17}$$

for each $i = 1, \ldots, d$. By the positivity of $\vec{a}$, the positivity of $p_2, \ldots, p_d$ is obvious. The value p_1 is also strictly positive because $0 < a_1 \leq p_1$. The sum of the elements of $\vec{p}$ is equal to $(d-1)(L+1)$ by definition. Hence $\vec{p} \in I_d((d-1)(L+1))$ holds. Because $(d-1)x_i/(m/2) < a_i \leq p_i$ it follows $\vec{a}$, $x_i < p_i(m/2)/(d-1)$ holds. This concludes the first inclusion. For the second inclusion, let $\vec{x} \in \mathbb{R}^d_{\geq 0}$ such that $x_i < p_i(m/2)/(d-1)$ for some $\vec{p} \in I_d((d-1)(L+1))$. Then $\sum_{i=1}^d x_i < (m/2)/(d-1) \sum_{i=1}^d p_i = (m/2)(L+1)$ as required. ■

By applying a linear transformation to the sets above and considering the intersection with $\mathbb{Z}^d_{\geq 0}$, we get the following corollary.

Corollary 4.3. Let $d > 1$, $L \in \mathbb{N}$ and $\vec{c} \in \mathbb{N}^d$. Then:

$$\begin{array}{rl} & \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid \sum_{i=1}^d c_i x_i \leq L \cdot (m/2) - 1 \,\} \\ \subseteq & \bigcup_{\vec{p} \in I_d((d-1)(L+1))} \bigcap_{j=1}^d \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid x_j \leq \lceil \frac{p_j \cdot (m/2)}{c_i (d-1)} \rceil - 1 \,\} \\ \subseteq & \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid \sum_{i=1}^d c_i x_i \leq (L+1) \cdot (m/2) - 1 \,\} \end{array}$$

Proof. By scaling the sets of lemma 4.2 by $\mathrm{diag}(1/c_1, \ldots, 1/c_d)$

$$\begin{array}{rl} & \{\, \vec{x} \in \mathbb{R}^d_{\geq 0} \mid \sum_{i=1}^d c_i x_i \leq L \cdot (m/2) - 1 \,\} \\ \subseteq & \bigcup_{\vec{p} \in I_d((d-1)(L+1))} \bigcap_{j=1}^d \{\, \vec{x} \in \mathbb{R}^d_{\geq 0} \mid x_j < \frac{p_j \cdot (m/2)}{c_i (d-1)} \,\} \\ \subseteq & \{\, \vec{x} \in \mathbb{R}^d_{\geq 0} \mid \sum_{i=1}^d c_i x_i < (L+1) \cdot (m/2) \,\} \end{array}$$

hence

$$\begin{array}{rl} & \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid \sum_{i=1}^d c_i x_i \leq L \cdot (m/2) - 1 \,\} \\ \subseteq & \bigcup_{\vec{p} \in I_d((d-1)(L+1))} \bigcap_{j=1}^d \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid x_j < \frac{p_j \cdot (m/2)}{c_i (d-1)} \,\} \\ \subseteq & \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid \sum_{i=1}^d c_i x_i < (L+1) \cdot (m/2) \,\} \end{array}$$

and the result follows. ■

By using this corollary many times, we get the sequence of inclusions by increasing L:

$$\begin{array}{l} \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid \sum_{i=1}^d c_i x_i \leq L \cdot (m/2) - 1 \,\} \\ \subseteq \bigcup_{\vec{p} \in I_d((d-1)(L+1))} \bigcap_{j=1}^d \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid x_j \leq \lceil \frac{p_j \cdot (m/2)}{c_i (d-1)} \rceil - 1 \,\} \\ \subseteq \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid \sum_{i=1}^d c_i x_i \leq (L+1) \cdot (m/2) - 1 \,\} \\ \subseteq \bigcup_{\vec{p} \in I_d((d-1)(L+2))} \bigcap_{j=1}^d \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid x_j \leq \lceil \frac{p_j \cdot (m/2)}{c_i (d-1)} \rceil - 1 \,\} \\ \subseteq \{\, \vec{x} \in \mathbb{Z}^d_{\geq 0} \mid \sum_{i=1}^d c_i x_i \leq (L+2) \cdot (m/2) - 1 \,\} \\ \ldots . \end{array}$$

This idea is used in the proofs of this chapter.

Based on Corollary 4.3, we define $\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b)$ for a linear inequality $\sum_{i=1}^d c_i x_i \leq b$. As the semantics of the inequality $x \leq b$ is different between the BV and the LIA when b is large, we also define a *reduced boxing* $\mathrm{box}_{\mathsf{BV}}(\vec{c}; b)$, the BV version of $\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b)$, to avoid the trouble of the difference of the semantics.

Definition 4.4. Let $\vec{c} \in \mathbb{N}^d$, $b \in \mathbb{N}$ and $L \in \mathbb{N}$ be the unique natural number such that $(L-1) \cdot (m/2) \leq b \leq L \cdot (m/2) - 1$. The *boxing* and *reduced boxing* of $\sum_{i=1}^d c_i x_i \leq b$ are formulae defined as follows:

$$\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b) \equiv \bigvee_{\vec{p} \in I_d((d-1)(L+1))} \bigwedge_{j=1}^d \left(x_j \leq \lceil \frac{p_j \cdot (m/2)}{c_j (d-1)} \rceil - 1 \right) \tag{4.18}$$

$$\mathrm{box}_{\mathsf{BV}}(\vec{c};b) \equiv \bigvee_{\vec{p}\in I_d((d-1)(L+1))} \bigwedge_{j=1}^{d} \left(x_j \leq \min\left(\lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1, m-1 \right) \right) \tag{4.19}$$

Given m and $b \in \mathbb{N}$, it is always possible to find a unique $L \in \mathbb{N}$ which satisfies Definition 4.4 by putting $L = \lfloor \frac{2b}{m} \rfloor + 1$. Then $L-1 = \lfloor \frac{2b}{m} \rfloor \leq \frac{2b}{m} < \lfloor \frac{2b}{m} \rfloor + 1 = L$ hence $(L-1)(m/2) \leq b < L(m/2)$ whence $(L-1)(m/2) \leq b \leq L(m/2) - 1$ because b and $L(m/2)$ are integral.

The following proposition asserts that the boxing and reduced boxing formulae share the same solution set when interpreted with, the LIA and BV semantics respectively.

Proposition 4.5. $\left[\!\left[\mathrm{box}_{\mathsf{LIA}}(\vec{c};b)\right]\!\right]_{\mathsf{LIA}} = \left[\!\left[\mathrm{box}_{\mathsf{BV}}(\vec{c};b)\right]\!\right]_{\mathsf{BV}}$

Proof. Put $L = \lfloor b/(n/2) \rfloor$. Let $\vec{x} \in \left[\!\left[\mathrm{box}_{\mathsf{LIA}}(\vec{c};b)\right]\!\right]_{\mathsf{LIA}}$ and $j \in \{1,\ldots,d\}$. Then there exists $\vec{p} \in I_d((d-1)(L+1))$ such that $x_j \leq \lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1$.

- Suppose $\lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1 \leq m-1$. Then $x_j \leq \lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1 = \min(\lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1, m-1) \bmod m$.
- Suppose $\lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1 \geq m$. Since $x_j \in \mathbb{M}$ it follows $x_j \leq m-1 = \min(\lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1, m-1) \bmod m$.

Thus $\vec{x} \in \left[\!\left[\mathrm{box}_{\mathsf{BV}}(\vec{c};b)\right]\!\right]_{\mathsf{BV}}$. Now let $\vec{x} \in \left[\!\left[\mathrm{box}_{\mathsf{BV}}(\vec{c};b)\right]\!\right]_{\mathsf{BV}}$ and $j \in \{1,\ldots,d\}$. There exists $\vec{p} \in I_d((d-1)(L+1))$ such that $x_j \leq \min(\lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1, m-1) \bmod m$.

- Suppose $\lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1 \leq m-1$. Then $x_j \leq \lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1$.
- Suppose $\lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1 \geq m$. Then $x_j \leq m-1 < \lceil \frac{p_j \cdot (m/2)}{c_j(d-1)} \rceil - 1$.

Therefore $\vec{x} \in \left[\!\left[\mathrm{box}_{\mathsf{LIA}}(\vec{c};b)\right]\!\right]_{\mathsf{LIA}}$. ■

Example 4.6. To demonstrate this equivalence, consider again $x + y \leq 3$ for $m = 8$. Then put $L = \lfloor 6/8 \rfloor + 1 = 1$ and $I_2((d-1)(L+1)) = I_2(2) = \{\langle 1,1\rangle\}$. Observe $\mathrm{box}_{\mathsf{LIA}}(\langle 1,1\rangle;3) = \mathrm{box}_{\mathsf{BV}}(\langle 1,1\rangle;3)$ since

$$\mathrm{box}_{\mathsf{LIA}}(\langle 1,1\rangle;3) = (x \leq \lceil 4/1 \rceil - 1 = 3) \wedge (y \leq \lceil 4/1 \rceil - 1 = 3)$$
$$\mathrm{box}_{\mathsf{BV}}(\langle 1,1\rangle;3) = (x \leq \min(3,7) = 3) \wedge (y \leq \min(3,7) = 3)$$

Example 4.7. Although $\left[\!\left[\mathrm{box}_{\mathsf{LIA}}(\vec{c};b)\right]\!\right]_{\mathsf{LIA}} = \left[\!\left[\mathrm{box}_{\mathsf{BV}}(\vec{c};b)\right]\!\right]_{\mathsf{BV}}$, it does not necessarily follow that $\left[\!\left[\mathrm{box}_{\mathsf{LIA}}(\vec{c};b)\right]\!\right]_{\mathsf{LIA}} = \left[\!\left[\mathrm{box}_{\mathsf{LIA}}(\vec{c};b)\right]\!\right]_{\mathsf{BV}}$. To illustrate this fact, consider $x + y \leq 7$ for $d = 2$ and $m = 4$. Thus $\vec{c} = \langle 1,1\rangle$ and $b = 7$. Then $L = \lfloor 14/4 \rfloor + 1 = 4$ and $I_2((d-1)(L+1)) = I_2(5) = \{\langle 1,4\rangle, \langle 2,3\rangle, \langle 3,2\rangle, \langle 4,1\rangle\}$ hence

$$\begin{aligned} \mathrm{box}_{\mathsf{LIA}}(\vec{c};b) \quad = \quad & (x \leq 1 \wedge y \leq 7) \vee (x \leq 3 \wedge y \leq 5) \vee \\ & (x \leq 5 \wedge y \leq 3) \vee (x \leq 7 \wedge y \leq 1) \end{aligned}$$

Therefore $\left[\!\left[\mathrm{box}_{\mathsf{LIA}}(\vec{c};b)\right]\!\right]_{\mathsf{LIA}} = \mathbb{M}^2$ but $(2,2) \notin \left[\!\left[\mathrm{box}_{\mathsf{LIA}}(\vec{c};b)\right]\!\right]_{\mathsf{BV}}$.

The following lemma shows that the solution sets for boxing grow monotonically as the constant of the inequality is relaxed.

Lemma 4.8. If $b \leq b'$ then $[\![\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b)]\!]_{\mathsf{LIA}} \subseteq [\![\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b')]\!]_{\mathsf{LIA}}$.

Proof. Let $L = \lfloor \frac{2b}{m} \rfloor + 1$ and $L' = \lfloor \frac{2b'}{m} \rfloor + 1$. Then $L \leq L'$.

- Suppose $L = L'$. Then $[\![\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b)]\!]_{\mathsf{LIA}} = [\![\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b')]\!]_{\mathsf{LIA}}$.
- Suppose $L + 1 \leq L'$. Let $\vec{x} \in [\![\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b)]\!]_{\mathsf{LIA}}$. By the second inclusion of Corollary 4.3 $\sum_{i=1}^{d} c_i x_i \leq (L+1)(m/2) - 1 \leq L'(m/2) - 1$. By the first inclusion of Corollary 4.3 it then follows $\vec{x} \in [\![\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b')]\!]_{\mathsf{LIA}}$.

■

The following theorem shows the special case of our boxing when b is small:

Theorem 4.9 (boxing without gapping)**.** *Let $\vec{c} \in \mathbb{N}^d$ and $b \in \mathbb{N}$. If $b < m/2$ then*

$$[\![\sum_{i=1}^{d} c_i x_i \leq b]\!]_{\mathsf{LIA}} = [\![(\sum_{i=1}^{d} c_i x_i \leq b) \wedge \mathrm{box}_{\mathsf{BV}}(\vec{c}; b)]\!]_{\mathsf{BV}}$$

Proof. Let $\vec{x} \in$ LHS. Since $\sum_{i=1}^{d} c_i x_i \leq b < m/2$ it follows

$$(\sum_{i=i}^{d} c_i x_i) \bmod m = \sum_{i=i}^{d} c_i x_i \leq b = b \bmod m$$

hence $\vec{x} \in [\![\sum_{i=1}^{d} c_i x_i \leq b]\!]_{\mathsf{BV}}$. By the first inclusion of Corollary 4.3 with $L = 1$, $\vec{x} \in [\![\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b)]\!]_{\mathsf{LIA}}$. By Proposition 4.5 $\vec{x} \in [\![\mathrm{box}_{\mathsf{BV}}(\vec{c}; b)]\!]_{\mathsf{BV}}$ hence

$$\vec{x} \in [\![\sum_{i=1}^{d} c_i x_i \leq b]\!]_{\mathsf{BV}} \cap [\![\mathrm{box}_{\mathsf{BV}}(\vec{c}; b)]\!]_{\mathsf{BV}} = (\mathrm{RHS}).$$

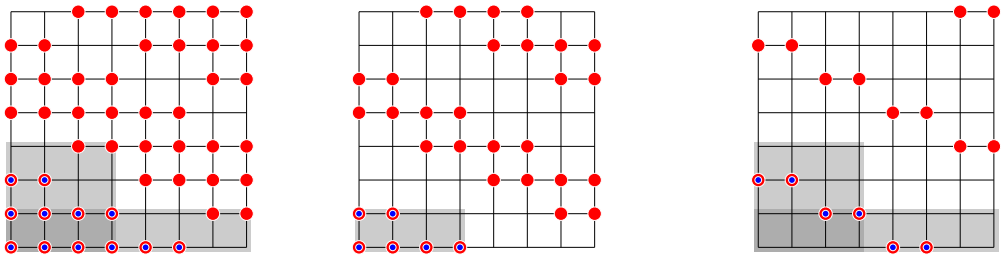
Let $\vec{x} \in$ RHS. Since $\vec{x} \in [\![\mathrm{box}_{\mathsf{BV}}(\vec{c}; b)]\!]_{\mathsf{LIA}}$ by Proposition 4.5 it follows $\vec{x} \in [\![\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b)]\!]_{\mathsf{LIA}}$. By the second part of inclusion of Corollary 4.3 with $L = 1$

$$\sum_{i=i}^{d} c_i x_i \leq (L+1) \cdot (m/2) - 1 < m.$$

hence $(\sum_{i=i}^{d} c_i x_i) \bmod m = \sum_{i=i}^{d} c_i x_i$. Moreover $b \bmod m = b$ since $b < m/2$. Because $\vec{x} \in [\![\sum_{i=1}^{d} c_i x_i \leq b]\!]_{\mathsf{BV}}$ it follows

$$\sum_{i=i}^{d} c_i x_i = (\sum_{i=i}^{d} c_i x_i) \bmod m \leq b \bmod m = b$$

hence $\vec{x} \in$ LHS. ■



(a) $x + 2y \leq 5$ with boxes (b) $x + 2y \leq 3$ with box (c) $0 \leq x + 2y - 4 \leq 1$ with boxes

Figure 4.2: Gapping and boxing for $x + 2y \leq 5$

Observe that the result requires $b < m/2$. In this circumstance $L = \lfloor 2b/m \rfloor + 1 = 1$ and the number of logical connectives in $\text{box}_{\mathsf{BV}}(\vec{c}; b)$ is determined by the cardinality of the set $I_d((d-1)(L+1)) = I_d(2(d-1))$, which is given below:

d	$2(d-1)$	$I_d(2(d-1))$	$\lvert I_d(2(d-1))\rvert$
2	2	$\Pi(\langle 1, 1\rangle)$	1
3	4	$\Pi(\langle 1, 1, 2\rangle)$	3
4	6	$\Pi(\langle 1, 1, 1, 3\rangle) \cup \Pi(\langle 1, 1, 2, 2\rangle)$	10
5	8	$\Pi(\langle 1, 1, 1, 1, 4\rangle) \cup \Pi(\langle 1, 1, 1, 2, 3\rangle) \cup \Pi(\langle 1, 1, 2, 2, 2\rangle)$	35

where $\Pi(v)$ denote the set of permutations of the vector v. For $d = 4$, $\text{box}_{\mathsf{BV}}(\vec{c}; b)$ thus requires $10(d-1) = 30$ binary conjunctions and $10 - 1 = 9$ disjunctions.

One might think that this boxing is still inefficient, since the number of boxes increases as d and L increase. The cardinality of $I_d((d-1)(L+1))$ in Definition 4.4 is bounded by $((d-1)(L+1) - 1)^{d-1}$, and since boxing is used for interpolants and d is the number of variables in the interpolant, d is expected to be small in practice (remember that the variables in an interpolant are the intersection of the variables in the target formulae). Also, if L is large, $\text{box}_{\mathsf{BV}}(\vec{c}; b)$ is small in practice because redundant boxes are omitted by the min in the definition of the reduced boxing. From the above two points, the number of resulting boxes is not so large in practice.

4.3.2 Boxing and Gapping

Example 4.10. Consider $[\![x + 2y \leq 5]\!]_{\mathsf{BV}}$ and $[\![x + 2y \leq 5]\!]_{\mathsf{LIA}}$ for $m = 8$ as shown in Figure 4.2(a). Observe

$$\text{box}_{\mathsf{BV}}(\langle 1, 2\rangle; 5) = (x \leq 3 \wedge y \leq 3) \vee (x \leq 7 \wedge y \leq 1)$$

which is illustrated by the two grey rectangles. Hence $\langle 2, 3\rangle \notin [\![x + 2y \leq 5]\!]_{\mathsf{LIA}}$ but $\langle 2, 3\rangle \in [\![x + 2y \leq 5 \wedge \text{box}_{\mathsf{BV}}(\langle 1, 2\rangle; 5)]\!]_{\mathsf{BV}}$ therefore using boxing alone is not sufficient to encode the

LIA inequality $x + 2y \leq 5$. This example means that applying boxing from Theorem 4.9 ignoring the assumption $b < m/2$ does not work.

Example 4.11. Yet the LIA inequality $x + 2y \leq 5$ can be decomposed as follows:

$$\begin{array}{rcl} [\![x + 2y \leq 5]\!]_{\mathsf{LIA}} & = & [\![x + 2y \leq 3]\!]_{\mathsf{LIA}} \cup [\![4 \leq x + 2y \leq 5]\!]_{\mathsf{LIA}} \\ & = & [\![x + 2y \leq 3]\!]_{\mathsf{LIA}} \cup [\![0 \leq x + 2y - 4 \leq 1]\!]_{\mathsf{LIA}} \end{array}$$

Figures 4.2(b, c) illustrates boxing for $x + 2y \leq 3$ and $0 \leq x + 2y - 4 \leq 1$ where:

$$\begin{array}{rcl} [\![x + 2y \leq 3]\!]_{\mathsf{LIA}} & = & [\![x + 2y \leq 3 \wedge \mathrm{box}_{\mathsf{BV}}(\langle 1, 2\rangle; 3)]\!]_{\mathsf{BV}} \\ & = & [\![x + 2y \leq 3 \wedge (x \leq 3 \wedge y \leq 1)]\!]_{\mathsf{BV}} \end{array}$$

Observe from Figure 4.2(c) that

$$[\![0 \leq x + 2y - 4 \leq 1]\!]_{\mathsf{LIA}} = [\![0 \leq x + 2y - 4 \leq 1]\!]_{\mathsf{BV}} \cap [\![\mathrm{box}_{\mathsf{BV}}(\langle 1, 2\rangle; 5)]\!]_{\mathsf{BV}}$$

and moreover $0 \bmod 8 = 0 \leq (x + 2y - 4) \bmod 8$ for all $(x, y) \in \mathbb{M}^2$ thus

$$[\![0 \leq x + 2y - 4 \leq 1]\!]_{\mathsf{LIA}} = [\![x + 2y - 4 \leq 1 \wedge \mathrm{box}_{\mathsf{BV}}(\langle 1, 2\rangle; 5)]\!]_{\mathsf{BV}}$$

therefore cumulatively $[\![x + 2y \leq 5]\!]_{\mathsf{LIA}} = [\![\varphi_1 \vee \varphi_2]\!]_{\mathsf{BV}}$ where

$$\begin{array}{rcllll} \varphi_1 & = & [x + 2y \leq 3 & \wedge & (x \leq 3 \wedge y \leq 1)] \\ \varphi_2 & = & [x + 2y - 4 \leq 1 & \wedge & ((x \leq 3 \wedge y \leq 3) \vee (x \leq 7 \wedge y \leq 1))] \end{array}$$

The general rule of the separation of the given inequality and the boxing is shown in this theorem:

Theorem 4.12 (boxing with gapping)**.** *Let* $\vec{c} \in \mathbb{N}^d$ *and* $b \in \mathbb{N}$. $[\![\sum_{i=1}^d c_i x_i \leq b]\!]_{\mathsf{LIA}} = [\![\phi_0 \vee \phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$ *where* $S = \lfloor b/(m/2) \rfloor$ *and*

$$\begin{array}{rcllll} \phi_0 & \equiv & \left(\sum_{i=1}^d c_i x_i - (S-2)(m/2) \leq m/2 - 1\right) & \wedge & \mathrm{box}_{\mathsf{BV}}(\vec{c}; (S-1)(m/2) - 1) \\ \phi_1 & \equiv & \left(\sum_{i=1}^d c_i x_i - (S-1)(m/2) \leq m/2 - 1\right) & \wedge & \mathrm{box}_{\mathsf{BV}}(\vec{c}; S(m/2) - 1) \\ \phi_2 & \equiv & \left(\sum_{i=1}^d c_i x_i - S(m/2) \leq b \bmod (m/2)\right) & \wedge & \mathrm{box}_{\mathsf{BV}}(\vec{c}; b) \end{array}$$

Proof. Let $\vec{x} \in$ LHS. Put $S' = \lfloor \sum_{i=1}^d c_i x_i/(m/2) \rfloor$ and observe $S' \leq S$.

- Suppose $S' < S$ and $S \equiv S' \mod 2$. Then there exists $n \in \mathbb{N}$ such that $S = S' + 2n$.

$$\begin{array}{rcl} \left[\sum_{i=1}^d c_i x_i - (S-2)(m/2)\right] \bmod m & = & \left[\sum_{i=1}^d c_i x_i - S'(m/2) - nm + m\right] \bmod m \\ & = & \left[\sum_{i=1}^d c_i x_i - S'(m/2)\right] \bmod m \end{array}$$

  But recall $S' = \lfloor \sum_{i=1}^d c_i x_i/(m/2) \rfloor$ hence $\sum_{i=1}^d c_i x_i - S'(m/2) < m/2$ and therefore $\vec{x} \in [\![\sum_{i=1}^d c_i x_i - (S-2)(m/2) \leq m/2 - 1]\!]_{\mathsf{BV}}$. But observe $(\sum_{i=1}^d c_i x_i)/(m/2) <$

$\lfloor(\sum_{i=1}^{d} c_i x_i)/(m/2)\rfloor + 1$ therefore

$$\begin{aligned}\sum_{i=1}^{d} c_i x_i &< (\lfloor(\sum_{i=1}^{d} c_i x_i)/(m/2)\rfloor + 1)(m/2)\\ &= (S' + 1)(m/2)\\ &\leq (S-1)(m/2).\end{aligned}$$

Thus $\sum_{i=1}^{d} c_i x_i \leq (S-1)(m/2) - 1$. By first inclusion of Collorary 4.3 with $L = S - 1$ and Proposition 4.5, $\vec{x} \in [\![\text{box}_{\mathsf{BV}}(\vec{c}; (S-1)\cdot(m/2) - 1)]\!]_{\mathsf{BV}}$. Hence $\vec{x} \in [\![\phi_0]\!]_{\mathsf{BV}}$.

- Suppose $S' < S$ and $S \equiv S' + 1 \mod 2$. Then there exists $n \in \mathbb{N}$ such that $S = S' + 2n + 1$.

$$\begin{aligned}\left[\sum_{i=1}^{d} c_i x_i - (S-1)(m/2)\right] \bmod m &= \left[\sum_{i=1}^{d} c_i x_i - S'(m/2) - nm\right] \bmod m\\ &= \left[\sum_{i=1}^{d} c_i x_i - S'(m/2)\right] \bmod m\end{aligned}$$

Recall that $S' = \lfloor\sum_{i=1}^{d} c_i x_i/(m/2)\rfloor$ hence $\sum_{i=1}^{d} c_i x_i - S'(m/2) < m/2$ and therefore $\vec{x} \in [\![\sum_{i=1}^{d} c_i x_i - (S-1)(m/2) \leq m/2 - 1]\!]_{\mathsf{BV}}$. But again

$$\begin{aligned}\sum_{i=1}^{d} c_i x_i &< (\lfloor(\sum_{i=1}^{d} c_i x_i)/(m/2)\rfloor + 1)(m/2)\\ &= (S' + 1)(m/2)\\ &\leq S(m/2).\end{aligned}$$

Thus $\sum_{i=1}^{d} c_i x_i \leq S(m/2) - 1$. By first inclusion of Corollary 4.3 with $L = S$ and Proposition 4.5, $\vec{x} \in [\![\text{box}_{\mathsf{BV}}(\vec{c}; S\cdot(m/2) - 1)]\!]_{\mathsf{BV}}$. Hence $\vec{x} \in [\![\phi_1]\!]_{\mathsf{BV}}$.

- Suppose $S = S'$. Since $\sum_{i=1}^{d} c_i x_i - S(m/2) = \sum_{i=1}^{d} c_i x_i - S'(m/2) = (\sum_{i=1}^{d} c_i x_i) \bmod (m/2)$, it follows $\sum_{i=1}^{d} c_i x_i - S(m/2) \leq b \bmod (m/2) < m$.
  Hence $\vec{x} \in [\![\sum_{i=1}^{d} c_i x_i - S(m/2) \leq b \bmod (m/2)]\!]_{\mathsf{BV}}$. By first inclusion of Collorary 4.3 with $L = \lfloor b/(m/2)\rfloor + 1$ and Proposition 4.5, $\vec{x} \in \text{box}_{\mathsf{BV}}(\vec{c}; b)$ from which it follows $\vec{x} \in [\![\phi_2]\!]_{\mathsf{BV}}$.

These three cases prove that $\vec{x} \in$ RHS. For the other direction, let $\vec{x} \in$ RHS.

- Suppose $\vec{x} \in [\![\phi_0]\!]_{\mathsf{BV}}$. By Proposition 4.5, $\vec{x} \in [\![\text{box}_{\mathsf{LIA}}(\vec{c}; (S-1)\cdot(m/2) - 1)]\!]_{\mathsf{LIA}}$. By the second inclusion of collorary 4.3 with $L = S - 1$ it follows

$$\sum_{i=1}^{d} c_i x_i \leq S\cdot(m/2) - 1 = \lfloor b/(m/2)\rfloor \cdot (m/2) - 1 \leq b$$

- Suppose $\vec{x} \in [\![\phi_1]\!]_{\mathsf{BV}}$. By Proposition 4.5, $\vec{x} \in [\![\text{box}_{\mathsf{LIA}}(\vec{c}; S\cdot(m/2) - 1)]\!]_{\mathsf{LIA}}$ and $\sum_{i=1}^{d} c_i x_i \leq (S+1)\cdot(m/2) - 1$ by the second inclusion of Corollary 4.3 with $L = S$. Because $\vec{x} \in [\![\sum_{i=1}^{d} c_i x_i - (S-1)(m/2) \leq m/2 - 1]\!]_{\mathsf{BV}}$ there exists $n \in \mathbb{Z}$ such that

$$0 \leq \sum_{i=1}^{d} c_i x_i - (S-1)(m/2) - mn \leq m/2 - 1$$

hence

$$(S+2n-1)(m/2) \leq \sum_{i=1}^{d} c_i x_i \leq (S+2n)(m/2) - 1$$

By combining inequalities $(S+2n-1)(m/2) \leq \sum_{i=1}^{d} c_i x_i \leq (S+1)\cdot(m/2) - 1$ hence $(2n-1)(m/2) < m/2$ from which it follows that $n \leq 0$. Therefore

$$\sum_{i=1}^{d} c_i x_i \leq S \cdot (m/2) - 1 = \lfloor b/(m/2) \rfloor \cdot (m/2) - 1 \leq b.$$

- Suppose $\vec{x} \in [\![\phi_2]\!]_{\mathsf{BV}}$. By Proposition 4.5 it follows $\vec{x} \in [\![\mathrm{box}_{\mathsf{LIA}}(\vec{c}; b)]\!]_{\mathsf{LIA}}$. Since $b < (S+1)(m/2)$, $\sum_{i=1}^{d} c_i x_i \leq (S+2)(m/2) - 1$ by the second inclusion of Collorary 4.3 with $L = S+1$. Since $\vec{x} \in [\![\sum_{i=1}^{d} c_i x_i - S(m/2) \leq b \bmod (m/2)]\!]_{\mathsf{BV}}$ there exists $n \in \mathbb{Z}$ such that

$$0 \leq \sum_{i=1}^{d} c_i x_i - S(m/2) - mn < b \bmod (m/2)$$

hence

$$(S+2n)(m/2) \leq \sum_{i=1}^{d} c_i x_i \leq b \bmod (m/2) + (S+2n)(m/2)$$

By combining inequalities $(S+2n)(m/2) \leq \sum_{i=1}^{d} c_i x_i \leq (S+2)(m/2) - 1$ hence $nm < m$ from which it follows $n \leq 0$ hence

$$\sum_{i=1}^{d} c_i x_i \leq b \bmod (m/2) + S \cdot (m/2)$$

But $b \bmod (m/2) + S \cdot (m/2) = b \bmod (m/2) + \lfloor b/(m/2) \rfloor \cdot (m/2) = b$ therefore $\sum_{i=1}^{d} c_i x_i \leq b$.

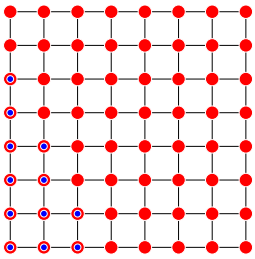
These three cases prove that $\vec{x} \in \mathrm{LHS}$. ■

Corollary 4.13 (boxing and gapping with simplification)**.** *If* $\lfloor b/(m/2) \rfloor = 1$ *or* $b \bmod m = m/2 - 1$ *then* $[\![\sum_{i=1}^{d} c_i x_i \leq b]\!]_{\mathsf{LIA}} = [\![\phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$.

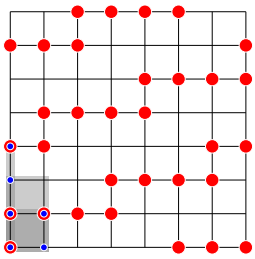
Proof. Let $S = \lfloor b/(m/2) \rfloor$.

- Suppose $S = 1$. Then $(S-1)\cdot(m/2) - 1 < 0$, $L \leq 0$ thus $I_d((d-1)(L+1)) = \emptyset$ and $[\![\phi_0]\!]_{\mathsf{BV}} = \emptyset$ hence $[\![\phi_0 \vee \phi_1 \vee \phi_2]\!]_{\mathsf{BV}} = [\![\phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$.
- Suppose $S \geq 2$. By theorem 4.12 $[\![\sum_{i=1}^{d} c_i x_i \leq b]\!]_{\mathsf{LIA}} = [\![\phi_0 \vee \phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$. Now consider

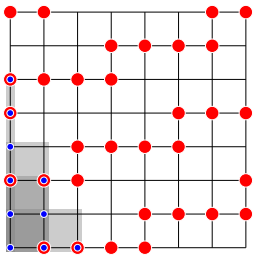
$$\phi_2 = \left(\sum_{i=1}^{d} c_i x_i - S(m/2) \leq b \bmod (m/2) \right) \wedge \mathrm{box}_{\mathsf{BV}}(\vec{c}; b)$$



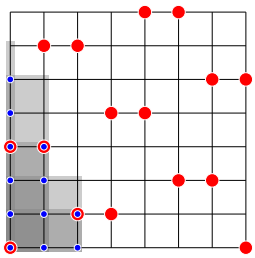
(a) $7x + 3y \leq 17$



(b) $\phi_0 = 7x + 3y - 8 \leq 3 \wedge \text{box}_{\mathsf{BV}}(\vec{c}; 11)$



(c) $\phi_1 = 7x + 3y - 12 \leq 3 \wedge \text{box}_{\mathsf{BV}}(\vec{c}; 15)$



(d) $\phi_2 = 7x + 3y - 16 \leq 1 \wedge \text{box}_{\mathsf{BV}}(\vec{c}; 17)$

Figure 4.3: Gapping and boxing for $7x + 3y \leq 17$ where $\vec{c} = \langle 7, 3 \rangle$, $m = 8$ and $S = 4$

Since $b \bmod m = m/2 - 1$ it follows $b \bmod (m/2) = m/2 - 1$. Since $S \geq 2$, $\sum_{i=1}^{d} c_i x_i - S(m/2) \mod m = \sum_{i=1}^{d} c_i x_i - (S-2)(m/2)$. Therefore

$$[\![\sum_{i=1}^{d} c_i x_i - S(m/2) \leq b \bmod (m/2)]\!]_{\mathsf{BV}} = [\![\sum_{i=1}^{d} c_i x_i - (S-2)(m/2) \leq m/2 - 1]\!]_{\mathsf{BV}}$$

Since $S = \lfloor b/(m/2) \rfloor$, $S(m/2) \leq b$ hence $(S-1)(m/2) - 1 \leq S(m/2) \leq b$. By lemma 4.8 $[\![\text{box}_{\mathsf{BV}}(\vec{c}; (S-1) \cdot (m/2) - 1)]\!]_{\mathsf{LIA}} \subseteq [\![\text{box}_{\mathsf{BV}}(\vec{c}; b)]\!]_{\mathsf{LIA}}$ and by proposition 4.5 $[\![\text{box}_{\mathsf{BV}}(\vec{c}; (S-1) \cdot (m/2) - 1)]\!]_{\mathsf{BV}} \subseteq [\![\text{box}_{\mathsf{BV}}(\vec{c}; b)]\!]_{\mathsf{BV}}$. Thus $[\![\phi_0]\!]_{\mathsf{BV}} \subseteq [\![\phi_2]\!]_{\mathsf{BV}}$ hence $[\![\phi_0 \vee \phi_1 \vee \phi_2]\!]_{\mathsf{BV}} = [\![\phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$

■

Example 4.14. Let $m = 8$ and consider again $x + 2y \leq 5$ so that $\vec{c} = \langle 1, 2 \rangle$. Then $S = \lfloor 5/4 \rfloor = 1$ and, applying corollary 4.13, $[\![x + 2y \leq 5]\!]_{\mathsf{LIA}} = [\![\phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$ where

$$\begin{array}{lcllcl} \phi_1 & \equiv & (x + 2y - 0 \cdot 4 \leq 4 - 1) & \wedge & \text{box}_{\mathsf{BV}}(\vec{c}; 1 \cdot 4 - 1) & = \varphi_1 \\ \phi_2 & \equiv & (x + 2y - 1 \cdot 4 \leq 5 \bmod 4) & \wedge & \text{box}_{\mathsf{BV}}(\vec{c}; 5) & = \varphi_2 \end{array}$$

aligning with the intuition given in Example 4.11.

Example 4.15. Figure 4.3 illustrates Theorem 4.12 for $7x + 3y \leq 17$ and $m = 8$. Then

$S = \lfloor 17/(8/2) \rfloor = 4$ and $[\![7x + 3y \leq 17]\!]_{\mathsf{LIA}} = [\![\phi_0 \vee \phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$ where

$$\begin{array}{rcl} \phi_0 & = & 7x + 3y - \ 8 \leq 3 \wedge \mathrm{box}_{\mathsf{BV}}(\vec{c}; 11) \\ \phi_1 & = & 7x + 3y - 12 \leq 3 \wedge \mathrm{box}_{\mathsf{BV}}(\vec{c}; 15) \\ \phi_2 & = & 7x + 3y - 16 \leq 1 \wedge \mathrm{box}_{\mathsf{BV}}(\vec{c}; 17) \end{array}$$

The $\mathrm{box}_{\mathsf{BV}}(\vec{c}; 11), \mathrm{box}_{\mathsf{BV}}(\vec{c}, 15), \mathrm{box}_{\mathsf{BV}}(\vec{c}; 17)$ formulae are again depicted in grey. For example,

$$\mathrm{box}_{\mathsf{BV}}(\vec{c}; 11) = (x \leq 0 \wedge y \leq 3) \vee (x \leq 1 \wedge y \leq 2) \vee (x \leq 1 \wedge y \leq 1)$$

because $d = 2$, $L = 3$ and $I_2((d-1)(L+1)) = \{\langle 1, 3\rangle, \langle 2, 2\rangle, \langle 3, 1\rangle\}$. From Figure 4.3 observe $[\![7x + 3y \leq 17]\!]_{\mathsf{LIA}} = [\![\phi_0]\!]_{\mathsf{BV}} \cup [\![\phi_1]\!]_{\mathsf{BV}} \cup [\![\phi_2]\!]_{\mathsf{BV}}$.

Example 4.16. Consider again example 4.14 where $S = 1$. Then $\phi_0 = \mathit{false}$ because $\mathrm{box}_{\mathsf{BV}}(\vec{c}; (S-1)(m/2) - 1) = \mathrm{box}_{\mathsf{BV}}(\vec{c}; -1) = \mathit{false}$. This is because $L = 0$ and $I_d((d-1)(L+1)) = I_2(1) = \emptyset$. Theorem 4.12 then gives $[\![x + 2y \leq 5]\!]_{\mathsf{LIA}} = [\![\phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$ which squares with Corollary 4.13.

4.3.3 Boxing, Gapping and Flipping

In order to apply the boxing and gapping above to linear inequalities with negative coefficients, we introduce a new technique called *flipping*. Flipping is defined for inequalities syntactically and semantically, and they are compatible in a compatible manner. The idea to apply boxing and gapping for general inequalities is to (1) apply flipping to the target linear inequality so that the signs of negative coefficients are inverted, (2) apply boxing and gapping to the inequality made by (1), and (3) apply the same flipping as (1) to the result of (2). To detail the transformation, we assume without loss of generality, that an inequality takes the syntactic form $\vec{c}^{+} \cdot \vec{x}^{+} + \vec{c}^{-} \cdot \vec{x}^{-} \leq b$ where $\vec{c}^{+} > \vec{0}$ and $\vec{c}^{-} < \vec{0}$. Hence $\vec{x} = \vec{x}^{+} \circ \vec{x}^{-}$ where $\circ$ denotes vector concatenation. The act of flipping reflects the solutions of the inequality simultaneously around the axes $x_1^{-} = (m-1)/2, \ldots, x_e^{-} = (m-1)/2$ where $\vec{x}^{-} = \langle x_1^{-}, \ldots, x_e^{-}\rangle$ and e is the dimension of x^{-}. We define the semantic flipping first:

Definition 4.17 (semantic flipping)**.** Given $e \in \{1, \ldots, d\}$, then the (semantic) flipping function $F_e : \mathbb{M}^d \to \mathbb{M}^d$ is defined:

$$F_e(\langle x_1^{+}, \ldots, x_{d-e}^{+}, x_1^{-}, \ldots, x_e^{-}\rangle) = \langle x_1^{+}, \ldots, x_{d-e}^{+}, m - 1 - x_1^{-}, \ldots, m - 1 - x_e^{-}\rangle.$$

Next, we define the syntactic flipping:

Definition 4.18. Given a partition of $\vec{x}$ into the sub-vectors $\vec{x}^{+} = \langle x_1^{+}, \ldots, x_{d-e}^{+}\rangle$ and $\vec{x}^{-} =$

$\langle x_1^-, \ldots, x_e^- \rangle$, then the (syntactic) flipping function $F_{\vec{x}^-}$ is defined:

$$\begin{array}{rcl} F_{\vec{x}^-}(\vec{c}^+ \cdot \vec{x}^+ + \vec{c}^- \cdot \vec{x}^- \leq b) & = & \vec{c}^+ \cdot \vec{x}^+ - \vec{c}^- \cdot \vec{x}^- + (m-1)(\vec{c}^- \cdot \vec{1}) \leq b \\ F_{\vec{x}^-}(f_1 \vee f_2) & = & F_{\vec{x}^-}(f_1) \vee F_{\vec{x}^-}(f_2) \\ F_{\vec{x}^-}(f_1 \wedge f_2) & = & F_{\vec{x}^-}(f_1) \wedge F_{\vec{x}^-}(f_2) \\ F_{\vec{x}^-}(\neg f) & = & \neg F_{\vec{x}^-}(f) \end{array}$$

The semantic flipping and syntactic flipping are compatible in LIA and BV:

Proposition 4.19. If $|\vec{x}^-| = e$ then

- $[\![F_{\vec{x}^-}(f)]\!]_{\mathsf{LIA}} = F_e([\![f]\!]_{\mathsf{LIA}})$
- $[\![F_{\vec{x}^-}(f)]\!]_{\mathsf{BV}} = F_e([\![f]\!]_{\mathsf{BV}})$

We prepares one lemma to prove the proposition.

Lemma 4.20. If $X, Y \subseteq \mathbb{M}^d$ then $F_e(X) \setminus F_e(Y) = F_e(X \setminus Y)$

Proof for lemma 4.20. Observe $\vec{x} \in F_e(X) \setminus F_e(Y)$ if and only if $F_e^{-1}(\vec{x}) \in X \setminus Y$ if and only if $\vec{x} \in F_e(X \setminus Y)$. ■

Proof for proposition 4.19. Structural induction on the formula f is used to show $[\![F_{\vec{x}^-}(f)]\!]_{\mathsf{LIA}} = F_e([\![f]\!]_{\mathsf{LIA}})$.

- Suppose $f = \vec{c}^+ \cdot \vec{x}^+ + \vec{c}^- \cdot \vec{x}^- \leq b$. Let $\vec{x} \in [\![f]\!]_{\mathsf{LIA}}$, $\vec{x} = \vec{x}^+ \circ \vec{x}^-$ and $\vec{y} = F_e(\vec{x}^+ \circ \vec{x}^-)$. Put $\vec{y} = \vec{y}^+ \circ \vec{y}^-$ where $\vec{y}^+ = \vec{x}^+$ and $\vec{y}^- = (m-1)\vec{1} - \vec{x}^-$. Observe

  $$\begin{array}{rcl} \vec{c}^+ \cdot \vec{x}^+ + \vec{c}^- \cdot \vec{x}^- & = & \vec{c}^+ \cdot \vec{x}^+ + \vec{c}^- \cdot \vec{x}^- - \vec{c}^- \cdot (m-1)\vec{1} + (m-1)(\vec{c}^- \cdot \vec{1}) \\ & = & \vec{c}^+ \cdot \vec{x}^+ - \vec{c}^- \cdot ((m-1)\vec{1} - \vec{x}^-) + (m-1)(\vec{c}^- \cdot \vec{1}) \\ & = & \vec{c}^+ \cdot \vec{y}^+ - \vec{c}^- \cdot \vec{y}^- + (m-1)(\vec{c}^- \cdot \vec{1}) \end{array}$$

  Therefore $\vec{y} \in [\![F_{\vec{x}^-}(f)]\!]_{\mathsf{LIA}}$.
- Suppose $f = f_1 \vee f_2$. By induction $[\![F_{\vec{x}^-}(f_1)]\!]_{\mathsf{LIA}} = F_e([\![f_1]\!]_{\mathsf{LIA}})$ and likewise $[\![F_{\vec{x}^-}(f_1)]\!]_{\mathsf{LIA}} = F_e([\![f_1]\!]_{\mathsf{LIA}})$. Therefore

  $$\begin{array}{rcl} [\![F_{\vec{x}^-}(f)]\!]_{\mathsf{LIA}} & = & [\![F_{\vec{x}^-}(f_1)]\!]_{\mathsf{LIA}} \cup [\![F_{\vec{x}^-}(f_2)]\!]_{\mathsf{LIA}} \\ & = & F_e([\![f_1]\!]_{\mathsf{LIA}}) \cup F_e([\![f_2]\!]_{\mathsf{LIA}}) \\ & = & F_e(f) \end{array}$$
- Suppose $f = f_1 \wedge f_2$. Analogous to the previous case.

- Suppose $f = \neg f'$. By lemma 4.20 it follows

$$\begin{array}{rcl}
[\![F_{\vec{x}^-}(f)]\!]_{\mathsf{LIA}} & = & [\![\neg F_{\vec{x}^-}(f')]\!]_{\mathsf{LIA}} \\
& = & \mathbb{M}^d \setminus [\![F_{\vec{x}^-}(f')]\!]_{\mathsf{LIA}} \\
& = & \mathbb{M}^d \setminus F_e([\![f']\!]_{\mathsf{LIA}}) \\
& = & F_e(\mathbb{M}^d) \setminus F_e([\![f']\!]_{\mathsf{LIA}}) \\
& = & F_e(\mathbb{M}^d \setminus [\![f']\!]_{\mathsf{LIA}}) \\
& = & F_e([\![\neg f']\!]_{\mathsf{LIA}}) \\
& = & F_e([\![f]\!]_{\mathsf{LIA}})
\end{array}$$

The $[\![F_{\vec{x}^-}(f)]\!]_{\mathsf{BV}} = F_e([\![f]\!]_{\mathsf{BV}})$ case is analogous. ■

The following corollary shows the boxing and gapping with flipping.

Corollary 4.21. Suppose $\vec{c}^+ > \vec{0}$, $\vec{c}^- < \vec{0}$ and

$$[\![\vec{c}^+ \cdot \vec{x}^+ - \vec{c}^- \cdot \vec{x}^- \leq b + (1-m)(\vec{c}^- \cdot \vec{1})]\!]_{\mathsf{LIA}} = [\![\phi_0 \vee \phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$$

Then $[\![\vec{c}^+ \cdot \vec{x}^+ + \vec{c}^- \cdot \vec{x}^- \leq b]\!]_{\mathsf{LIA}} = [\![F_{\vec{x}^-}(\phi_0) \vee F_{\vec{x}^-}(\phi_1) \vee F_{\vec{x}^-}(\phi_2)]\!]_{\mathsf{BV}}$

Proof. Suppose

$$[\![\vec{c}^+ \cdot \vec{x}^+ - \vec{c}^- \cdot \vec{x}^- \leq b + (1-m)(\vec{c}^- \cdot \vec{1})]\!]_{\mathsf{LIA}} = [\![\phi_0 \vee \phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$$

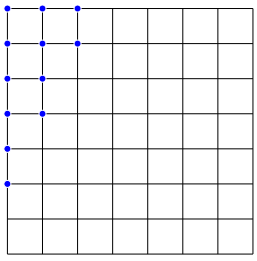
Then

$$\begin{array}{rcl}
[\![\vec{c}^+ \cdot \vec{x}^+ + \vec{c}^- \cdot \vec{x}^- \leq b]\!]_{\mathsf{LIA}} & = & F_e(F_e([\![\vec{c}^+ \cdot \vec{x}^+ + \vec{c}^- \cdot \vec{x}^- \leq b)]\!]_{\mathsf{LIA}})) \\
& = & F_e([\![F_{\vec{x}^-}(\vec{c}^+ \cdot \vec{x}^+ + \vec{c}^- \cdot \vec{x}^- \leq b)]\!]_{\mathsf{LIA}}) \\
& = & F_e([\![\vec{c}^+ \cdot \vec{x}^+ - \vec{c}^- \cdot \vec{x}^- + (m-1)(\vec{c}^- \cdot \vec{1}) \leq b]\!]_{\mathsf{LIA}}) \\
& = & F_e([\![\vec{c}^+ \cdot \vec{x}^+ - \vec{c}^- \cdot \vec{x}^- \leq b + (1-m)(\vec{c}^- \cdot \vec{1})]\!]_{\mathsf{LIA}}) \\
& = & F_e([\![\phi_0 \vee \phi_1 \vee \phi_2]\!]_{\mathsf{BV}}) \\
& = & [\![F_{\vec{x}^-}(\phi_0 \vee \phi_1 \vee \phi_2)]\!]_{\mathsf{BV}} \\
& = & [\![F_{\vec{x}^-}(\phi_0) \vee F_{\vec{x}^-}(\phi_1) \vee F_{\vec{x}^-}(\phi_2)]\!]_{\mathsf{BV}}
\end{array}$$

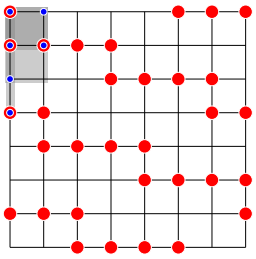
■

Example 4.22. Consider $\phi = 7x - 3y \leq -4$ which is illustrated in Figure 4.4(a). Then $\vec{x}^+ = \langle x \rangle$, $\vec{x}^- = \langle y \rangle$ and $F_{\vec{x}^-}(\phi) = F_{\langle y \rangle}(\phi) = 7x + 3y - 21 \leq -4$. Figure 4.3(a) shows $[\![7x + 3y - 21 \leq -4]\!]_{\mathsf{LIA}} = [\![7x + 3y \leq 17]\!]_{\mathsf{LIA}}$ and so building on example 4.15 $[\![7x + 3y \leq 17]\!]_{\mathsf{LIA}} = [\![\phi_0 \vee \phi_1 \vee \phi_2]\!]_{\mathsf{BV}}$. By corollary 4.21 it follows $[\![\phi]\!]_{\mathsf{LIA}} = [\![F_{\langle y \rangle}(\phi_0) \vee F_{\langle y \rangle}(\phi_1) \vee F_{\langle y \rangle}(\phi_2)]\!]_{\mathsf{BV}}$ where $F_{\langle y \rangle}(\phi_0)$, $F_{\langle y \rangle}(\phi_1)$ and $F_{\langle y \rangle}(\phi_2)$ are given in Figure 4.4(b), (c) and (d) respectively. Finally, to illustrate the handling of boxing, recall $\mathrm{box}_{\mathsf{BV}}(\vec{c}; 11)$ from example 4.15 and

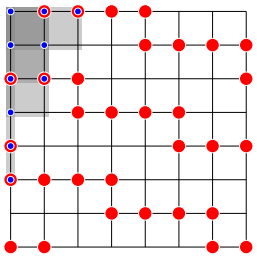
$$\begin{array}{rcl}
\mathrm{box}_{\mathsf{BV}}(\vec{c}; 11) & = & (x \leq 0 \wedge y \leq 3) \\
& \vee & (x \leq 1 \wedge y \leq 2) \\
& \vee & (x \leq 1 \wedge y \leq 1)
\end{array}
\qquad
\begin{array}{rcl}
F_{\langle y \rangle}(\mathrm{box}_{\mathsf{BV}}(\vec{c}; 11)) & = & (x \leq 0 \wedge (-y + 7 \leq 3)) \\
& \vee & (x \leq 1 \wedge (-y + 7 \leq 2)) \\
& \vee & (x \leq 1 \wedge (-y + 7 \leq 1))
\end{array}$$



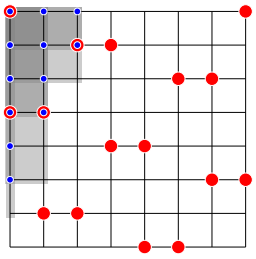
(a) $[\![\phi]\!]_{\mathsf{LIA}}$



(b) $\begin{array}{rcl} F_{\langle y\rangle}(\phi_0) & = & 7x - 3y + 13 \leq 3 \\ & \wedge & F_{\langle y\rangle}(\mathrm{box}_{\mathsf{BV}}(\vec{c}; 11)) \end{array}$



(c) $\begin{array}{rcl} F_{\langle y\rangle}(\phi_1) & = & 7x - 3y - 9 \leq 3 \\ & \wedge & F_{\langle y\rangle}(\mathrm{box}_{\mathsf{BV}}(\vec{c}; 15)) \end{array}$



(d) $\begin{array}{rcl} F_{\langle y\rangle}(\phi_2) & = & 7x - 3y + 5 \leq 1 \\ & \wedge & F_{\langle y\rangle}(\mathrm{box}_{\mathsf{BV}}(\vec{c}; 17)) \end{array}$

Figure 4.4: Flipping $\phi = 7x - 3y \leq -4$ where $m = 8$, $\vec{x} = \langle x, y\rangle$, $\vec{x}^+ = \langle x\rangle$ and $\vec{x}^- = \langle y\rangle$

Finally observe

$$\begin{array}{rcl} [\![x \leq 0 \wedge (-y + 7 \leq 3)]\!]_{\mathsf{LIA}} & = & \{(0, y) \in \mathbb{M}^2 \mid 4 \leq y \leq 7\} \\ [\![x \leq 1 \wedge (-y + 7 \leq 2)]\!]_{\mathsf{LIA}} & = & \{(x, y) \in \mathbb{M}^2 \mid 0 \leq x \leq 1 \wedge 5 \leq y \leq 7\} \end{array}$$

and that the disjunct $(x \leq 1 \wedge (-y + 7 \leq 1))$ is actually redundant.

4.3.4 Boxing, Gapping, Flipping and Demoding

In this section, we discuss how to translate back an LIA interpolant including the floor function into a BV interpolant. Griggio reported that formulae including the floor function like $-x_2 + x_3 - 256\lfloor -x_2/256\rfloor \leq 255$ occurs while computing LIA interpolants, and it is an obstacle to get a BV interpolant [52, Example 5]. As the technique of LIA interpolation can return formulae of the form $\vec{c} \cdot \vec{x} + n'\lfloor \vec{c}' \cdot \vec{x}/n\rfloor \leq b$ [28] or $\vec{c} \cdot \vec{x} + n'\lceil \vec{c}' \cdot \vec{x}/n\rceil \leq b$ [53], this obstacle is inevitable.

We experimentally observe that the divisors are the powers of two and the formulae are of

the form $\vec{c}\cdot\vec{x}+n'2^n\lfloor\vec{c}'\cdot\vec{x}/2^n\rfloor\leq b$, which is due to the wrap-around of the BV theory, and we propose a method of translating back such formulae. In this assumption, we can use the property $(\vec{c}\cdot\vec{x} \bmod 2^n) \bmod m = \vec{c}\cdot\vec{x} \bmod 2^n$ for $n \leq w$.

First we define the semantics for the formulae including the floor function:

Definition 4.23. *If* $\ell \equiv \vec{c}\cdot\vec{x}+n'\lfloor\vec{c}'\cdot\vec{x}/2^n\rfloor\leq b$ *then*

$$\begin{aligned}\llbracket\ell\rrbracket_{\mathsf{LIA}} &= \{\,\vec{x}\in\mathbb{M}^d \mid \vec{c}\cdot\vec{x}+n'\lfloor\vec{c}'\cdot\vec{x}/2^n\rfloor\leq b\,\}\\ \llbracket\ell\rrbracket_{\mathsf{BV}} &= \{\,\vec{x}\in\mathbb{M}^d \mid (\vec{c}\cdot\vec{x}+n'\lfloor\vec{c}'\cdot\vec{x}/2^n\rfloor) \bmod m \leq b \bmod m\,\}\end{aligned}$$

The following proposition gives the technique to translate back the formulae into the BV theory. The idea is (1) to rewrite the floor function with mod and transform the formulae so that it is supported by the BV theory, and (2) to replace the term including mod with a fresh variable y, translate the formula into a linear inequality with variables $\vec{x}, y$.

Proposition 4.24. Suppose $0\leq n\leq w$ and $\llbracket(\vec{c}+n'\vec{c}')\cdot\vec{x}-n'y\leq b\rrbracket_{\mathsf{LIA}} = \llbracket\phi\rrbracket_{\mathsf{BV}}$. If y does not occur in $\vec{x}$ then

$$\llbracket\vec{c}\cdot\vec{x}+n'2^n\lfloor\vec{c}'\cdot\vec{x}/2^n\rfloor\leq b\rrbracket_{\mathsf{LIA}} = \llbracket\phi[y\mapsto\vec{c}'\cdot\vec{x} \bmod 2^n]\rrbracket_{\mathsf{BV}}$$

Proof. Suppose $\llbracket(\vec{c}+n'\vec{c}')\cdot\vec{x}-n'y\leq b\rrbracket_{\mathsf{LIA}} = \llbracket\phi\rrbracket_{\mathsf{BV}}$ and that y does not occur in $\vec{x}$. Then

$$\begin{array}{ll}
\vec{x}\in\llbracket\vec{c}\cdot\vec{x}+n'2^n\lfloor\vec{c}'\cdot\vec{x}/2^n\rfloor\rrbracket_{\mathsf{LIA}} & \text{iff}\\
\vec{c}\cdot\vec{x}+n'2^n\lfloor\vec{c}'\cdot\vec{x}/2^n\rfloor\leq b & \text{iff}\\
\vec{c}\cdot\vec{x}+n'(\vec{c}'\cdot\vec{x}-(\vec{c}'\cdot\vec{x} \bmod 2^n))\leq b & \text{iff}\\
\exists y.(\vec{c}+n'\vec{c}')\cdot\vec{x}-n'y\leq b\wedge y=\vec{c}'\cdot\vec{x} \bmod 2^n & \text{iff}\\
\exists y.\vec{x}\cdot y\in\llbracket(\vec{c}+n'\vec{c}')\cdot\vec{x}-n'y\leq b\rrbracket_{\mathsf{LIA}}\wedge y=\vec{c}'\cdot\vec{x} \bmod 2^n & \text{iff}\\
\exists y.\vec{x}\cdot y\in\llbracket\phi\rrbracket_{\mathsf{BV}}\wedge y=\vec{c}'\cdot\vec{x} \bmod 2^n & \text{iff}\\
\vec{x}\in\llbracket\phi[y\mapsto\vec{c}'\cdot\vec{x} \bmod 2^n]\rrbracket_{\mathsf{BV}} &
\end{array}$$

■

Inequalities such as $\vec{c}\cdot\vec{x}+n'2^n\lceil\vec{c}'\cdot\vec{x}/2^n\rceil\leq b$ can be handled similarly. For completeness, we note that expansion can be applied for general inequalities including mod by the naive expansion, though it is not sufficient:

Proposition 4.25. Suppose $n>0$. Then

$$\llbracket\vec{c}\cdot\vec{x}+n'\lfloor\vec{c}'\cdot\vec{x}/n\rfloor\leq b\rrbracket_{\mathsf{LIA}} = \Big\llbracket\bigvee_{i=\ell}^{u}(\vec{c}\cdot\vec{x}\leq b-n'i\wedge ni\leq\vec{c}'\cdot\vec{x}\leq ni-1)\Big\rrbracket_{\mathsf{LIA}}$$

where $\ell=\min\{\lfloor\vec{c}'\cdot\vec{x}/n\rfloor \mid \vec{x}\in\mathbb{M}^d\}$ and $u=\max\{\lfloor\vec{c}'\cdot\vec{x}/n\rfloor \mid \vec{x}\in\mathbb{M}^d\}$.

Table 4.2: Comparison of the theories: performance and correctness

Theory	Safety	Solved	Time (seconds)	Size (inequalities)
LIA	safe	165	15.1	440
	unsafe	41	9.0	392
	(total)	206	13.9	431
BV (naive)	safe	87	30.1	32583
	unsafe	57	24.2	49138
	(total)	144	27.8	39136
BV (boxing)	safe	99	20.0	6938
	unsafe	66	20.1	15246
	(total)	165	20.0	10261

		LIA	
		safe	unsafe
BV	safe	90	1
	unsafe	17	34

4.4 Experiments

To evaluate the performance of boxing we implemented a model checker based on the lazy abstraction (IMPACT) [89] algorithm. The model checker is implemented in Python 3.7.2 and uses MathSAT5 [28] for satisfiability checking and interpolation over the LIA. The model checker parses a subset of the C language, but is rich enough to handle 312 benchmarks drawn from [8, 43]. The model checker was instantiated with: (1) the LIA interpolation [53]; (2) the BV interpolation by covering the solutions of an LIA interpolate with columns (recall f_2 of section 4.2); and (3) the BV interpolation by covering the solutions of an LIA interpolate using boxing, gapping and flipping. Experiments were performed using an Amazon Web Service EC2 c3.xlarge cloud architecture of 14 EC2 Computing Units [120] each equipped with 4 cores and 7.5 GB of RAM. The timeout for each run of IMPACT was set to 600 seconds.

Arithmetic is idealized in configuration (1) taking no account of integer overflow and underflow. This is not, in general, safe. In configurations (2) and (3) the model checker interprets machine arithmetic and bit operations using the LIA encoding of BV operations outlined in [52, Fig 1]. This is safe but complicates the LIA formulae, often substantially. One would expect this to enlarge the interpolants, even before boxing and gapping are deployed. We would also expect (1) to be substantially faster than (2) and (3). Due to differences in the semantics of arithmetic, we might also see differences in the number of programs proved to be safe or found to be unsafe. The experiments quantify these predictions. To discuss the experiments, (2) will be referred to as the naive encoding, even though it improves on complete enumeration (recall f_1 of section 4.2).

4.4.1 Overall Result

Table 4.2 summarises the outcomes of running IMPACT on all 312 programs, using the three different instances of interpolation, categorized as to whether the run proved the safety (safe rows) or found a counterexample (unsafe rows). The Solved column of the left-hand table gives the total of the programs that were either shown to be safe or unsafe within 600 seconds. Time is the mean execution of a run (for all those programs which did not timeout). Size is the mean of total number of atomic constraints in all interpolants encountered over a run (for those programs which did not timeout). We observe that more programs can be analyzed to completion with LIA than with BV, as one would expect, but that BV (boxing) improves on BV (naive), the speedup being significant when proving safety.

The right-hand table compares a terminating run of LIA to a terminating run of BV (boxing). For 17 of these 142 runs, LIA (incorrectly) verified the program to be safe whereas BV found a counter-example. Unexpectedly for `trex03_true-unreach-call.i.annot.c` from [43], LIA found a counter-example but BV verified safety. This program contains three integers, `x1`, `x2` and `x3`, which can become negative in the idealized arithmetic employed in LIA, triggering an assertion. But `x1`, `x2` and `x3` are actually unsigned.

4.4.2 Runtime for Naive Encoding and Boxing

The scatter plot of Figure 4.5 compares the runtime of the naive encoding against that of boxing and its allied techniques of gapping and flipping. The scatter plot excludes timeouts and depicts 151 pairs of runs. Almost all points are under the dotted line, indicating the boxing significantly improves performance. The line graph plots the ratio of the execution times, from which we observe that boxing does not accelerate the verification for almost half of the runs, but does speed it up between 2- and 256-fold for the other half.

4.4.3 Interpolant Size for Naive Encoding and Boxing

The line graph on Figure 4.6 compares the relative size of interpolants for boxing versus the naive encoding. Size is the sum of the sizes of all the interpolants generated during a run, where the size of an interpolant is itself defined as the number of atomic constraints that occur within it. We observe that the size ratio is around one for most problems, but a second peak occurs at 1/32, giving an overall size reduction. The scatter plot explores how interpolant size correlates with runtime, showing how the relative size of interpolants varies with relative runtimes. We observe that reducing the size of interpolants improves runtime, and that two

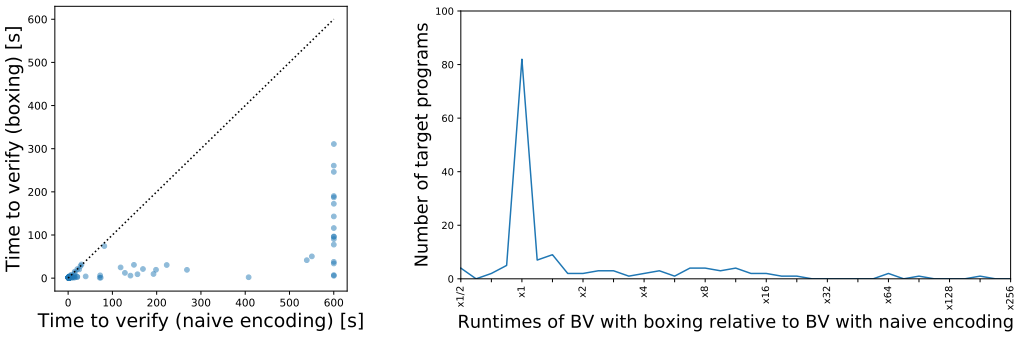


Figure 4.5: Runtime of boxing versus naive: scatter plot and ratio plot

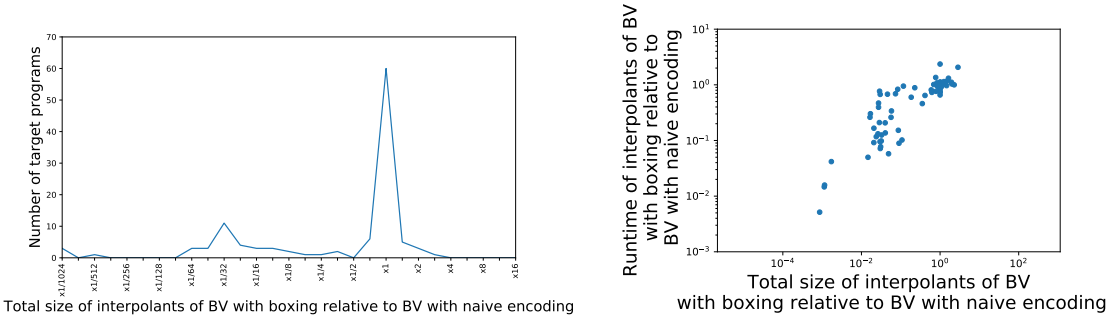


Figure 4.6: Size of interpolants in boxing versus naive and its impact on performance

peaks of the line graph manifesting themselves as two clusters of points in the scatter plot.

4.5 Conclusions

To repurpose efficient LIA interpolation engines to BV, we have shown how to systematically construct a BV formula so that its solutions are exactly those of an LIA interpolant. Since an LIA interpolant summarises the reason for a conflict between two LIA formulae, we seek to retain its compact structure by introducing no more than simple boxes around the LIA solutions which block extraneous BV solutions. When this encoding tactic, called boxing, is not applicable, gapping is used to decompose an LIA inequality into two or more inequalities which are amenable to boxing. We show how the size of the resulting BV interpolants are smaller than BV interpolants constructed by merely partitioning the LIA solutions into columns, and demonstrate how boxing and gapping improves the runtime of an interpolation-based model checker. We instantiate a model checker with LIA and BV to compare their performance, and conclude that with this encoding BV interpolation is feasible. Because of wrap-arounds,

the BV theory is substantially more complicated than the LIA for interpolation, yet the BV theory is no more than twice as slow as the LIA for over half the benchmarks. Furthermore, the resulting BV interpolants can be validated, independently of the LIA, just using a BV solver.

4.6 Future Work

Griggio originally proposed a multi-layered interpolation, which combines multiple methods to compute interpolants in the BV theory. The method using interpolation in the LIA theory was one layer of Griggio's work, and our work's aim was to improve this layer. Measuring the end-to-end performance of the multi-layered interpolation combined with our work, and investigating the effect of our work in total would be interesting.

Chapter 5

Weighted Automata Extraction from Recurrent Neural Networks via Regression on State Spaces

This chapter is based on joint work [103] with Masaki Waga, Taro Sekiyama, and Ichiro Hasuo.

5.1 Introduction

5.1.1 Background

Deep neural networks (DNNs) have been successfully applied to domains such as text, speech, and image processing. *Recurrent neural networks (RNNs)* [27, 62] is a class of DNNs equipped with the capability of processing sequential data of variable length. The great success of RNNs has been seen in machine translation [125], speech recognition [140], and anomaly detection [86, 136].

While it has been experimentally shown that RNNs are a powerful tool to process, predict, and model sequential data, there are known drawbacks in RNNs such as interpretability and costly inference. A research line that attacks this challenge is *automata extraction* [104, 134]. Focusing on RNNs' use as *acceptors* (i.e., receiving an input sequence and producing a single Boolean output), these pieces of works extract a *finite-state automaton* from an RNN as a succinct and interpretable surrogate. Automata extraction exposes internal transition between the states of an RNN in the form of an automaton, which is then amenable to algorithmic analyses such as reachability and model checking [9]. Automata extraction can also be seen as *model compression*: finite-state automata are usually more compact, and cheaper to run,

than neural networks.

5.1.2 Extracting WFAs from RNNs

Most of the existing automata extraction techniques target Boolean-output RNNs, which however excludes many applications. In sentiment analysis, it is desired to know the quantitative strength of sentiment, besides its (Boolean) existence [25]. RNNs with real values as their outputs are also useful in classification tasks. For example, predicting class probabilities is a key in some approaches to semi-supervised learning [139] and ensemble [19].

This motivates extraction of *quantitative* finite-state machines as abstraction of RNNs. We find the formalism of *weighted finite automata* (WFAs) suited for this purpose. A WFA is a finite-state machine—much like a deterministic finite automaton (DFA)—but its transitions as well as acceptance values are real numbers (instead of Booleans).

5.1.3 Contribution: Regression-Based WFA Extraction from RNNs

Our main contribution is a procedure that takes a (real-output) RNN R, and returns a WFA A_R that abstracts R. The procedure is based on the WFA learning algorithm in [12], that is in turn based on the famous L^* *algorithm* for learning DFAs [6]. These algorithms learn automata by a series of so-called *membership queries* and *equivalence queries*. In our procedure, a membership query is implemented by an inference of the given RNN R. We iterate membership queries and use their results to construct a WFA A.

The role of equivalence queries is to say when to stop this iteration: it asks if A and R are "equivalent," that is, if the WFA A obtained so far is comprehensive enough to cover all the possible behaviors of R. This is not possible in general—RNNs are more expressive than WFAs—therefore we inevitably resort to an approximate method. Our technical novelty lies in the method for answering equivalence queries; notably it uses a *regression* method—e.g., the Gaussian process regression (GPR) and the kernel ridge regression (KRR)—for abstraction of the state space of R.

We conducted experiments to evaluate the effectiveness of our approach. In particular, we are concerned with the following questions: 1) how similar the behavior of the extracted WFA A_R, and that of the original RNN R, are; 2) how applicable our method is to an RNN that is a more expressive model than WFAs; and 3) how efficient the inference of the WFA A_R is, when compared with the inference of R. The experiments we designed for the questions 1) and 3) are with RNNs trained using randomly generated WFAs. The results show, on the question 1), that the WFAs extracted by our method approximate the original RNNs

accurately. This is especially so when compared with a baseline algorithm (a straightforward adaptation of [12]). On the question 3), the inference of the extracted WFAs are about 1300 times faster than that of the original RNNs. On the question 2), we devised an RNN that models a weighted variant of a (non-regular) language of balanced parentheses. Although this weighted language is beyond WFAs' expressivity, we found that our method extracts WFAs that successfully approximate the RNN up-to a certain depth bound.

The chapter is organized as follows. Angluin's L* algorithm and its weighted adaptation are recalled in Section 5.2. Our WFA extraction procedure is described in Section 5.3 focusing on our novelty, namely the regression-based procedure for answering equivalence queries. Comparison with the DFA extraction by [134] is given there, too. In Section 5.4 we discuss our experiment results.

5.1.4 Potential Applications

A major potential application of WFA extraction is to analyze an RNN R via its interpretable surrogate A_R. The theory of WFAs offers a number of analysis methods, such as *bisimulation metric* [11]—a distance notion between WFAs—that will allow us to tell how apart two RNNs are.

Another major potential application is as "a poor man's RNN R." While RNN inference is recognized to be rather expensive (especially for edge devices), simpler models by WFAs should be cheaper to run. Indeed, our experimental results show (Section refsec:implExpr) that WFA inference is about 1300 times faster than inference of original RNNs.

Since WFAs are defined over a finite alphabet, we restrict to RNNs R that take sequences over a *finite* alphabet. This restriction should not severely limit the applicability of our method. Indeed, such RNNs (over a finite alphabet) have successfully applications in many domains, including intrusion prediction [86], malware detection [136], and DNA-protein binding prediction [122]. Moreover, even if inputs are real numbers, *quantization* is commonly employed without losing a lot of precision. See [54] for example.

5.1.5 Related Work

The relationship between RNNs and automata has been studied in both non-quantitative [93, 104, 134, 135] and quantitative [7, 113] settings. Some of them feature automata extraction from RNNs; we shall now discuss recent ones among them.

The work by [134] is a pioneer in automata extraction from RNNs. They extract DFAs from RNNs, using a variation of L*algorithm, much like this work. We provide a systematic

comparison in Section 5.3.3, identifying some notable similarities and differences.

[7] extract a WFA from a black-box sequence acceptor whose example is an RNN. Their method does not use equivalence queries; in contrast, we exploit the internal state space of an RNN to approximately answer equivalence queries.

DeepStellar [46] extracts Markov chains from RNNs, and uses them for coverage-based testing and adversarial sample detection. Their extraction, differently from our L^*-like method, uses profiling from the training data and discrete abstraction of the state space.

[133] propose a new RNN architecture that makes it easier to extract a DFA from a trained model. To apply their method, one has to modify the structure of an RNN before training, while our method does not need any special structure to RNNs and can be applied to already trained RNNs.

[119] introduce a neural network architecture that can represent (restricted forms of) CNNs and RNNs. WFAs could also be expressed by their architecture, but extraction of automata is out of their interest.

A major approach to optimizing neural networks is by compression: *model pruning* [57], *quantization* [54], and *distillation* [21]. Combination and comparison with these techniques is an interesting future work.

5.1.6 Organization of the Chapter

In Section 5.2, we introduce some preliminary definitions. In Section 5.3, we propose our algorithm and discuss the details. In Section 5.4, we give some research questions and the experimental result. Section 5.5 concludes the chapter, and in Section 5.6 we discuss the future work.

5.2 Preliminaries

We fix a finite alphabet Σ. The set of (finite-length) words over Σ is Σ^*. The *empty word* (of length 0) is denoted by ε. The length of a word $w \in \Sigma^*$ is denoted by $|w|$.

We recall basic notions on WFAs. See [45] for details.

Definition 5.1 (WFA)**.** *A* weighted finite automaton *(WFA) over Σ is a quadruple $A = \big(Q_A, \alpha_A, \beta_A, (A_\sigma)_{\sigma\in\Sigma}\big)$. Here Q_A is a finite set of* states*; α_A, β_A are row vectors of size $|Q_A|$ called the* initial *and* final *vectors; and A_σ is a* transition *matrix of σ, given for each $\sigma \in \Sigma$. For each $\sigma \in \Sigma$, A_σ is a matrix of size $|Q_A| \times |Q_A|$.*

Definition 5.2 (configuration of a WFA)**.** *Let A be the WFA in Definition 5.1. A* configu-

ration *of A is a row vector $x \in \mathbb{R}^{Q_A}$. For a word $w = \sigma_1\sigma_2\ldots\sigma_n \in \Sigma^*$ (where $\sigma_i \in \Sigma$), the* configuration *of A at w is defined by $\delta_A(w) = \alpha_A^\top \cdot \left(\prod_{i=1}^n A_{\sigma_i}\right)$.*

Obviously $\delta_A(w) \in \mathbb{R}^{Q_A}$ is a row vector of size $|Q_A|$; it records the weight at each state $q \in Q_A$ after reading w.

Definition 5.3 (weight $f_A(w)$ of a word in a WFA)**.** *Let a WFA A and a word $w = \sigma_1\ldots\sigma_n$ be as in Definition 5.2. The* weight *of w in A is given by $f_A(w) = \alpha_A^\top \cdot \left(\prod_{i=1}^n A_{\sigma_i}\right) \cdot \beta_A$, multiplying the final vector to the configuration at w.*

Example 5.4. *Let $\Sigma = \{a, b\}$, $Q_A = \{q_1, q_2, q_3\}$, $\alpha_A = (1\ 2\ 3)^\top$, $\beta_A = (0\ {-1}\ 1)^\top$, $A_a = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 0 & 0 \\ 0 & 4 & 0 \end{pmatrix}$, and $A_b = \begin{pmatrix} -1 & 1 & 0 \\ 0 & 3 & 0 \\ -2 & 4 & 0 \end{pmatrix}$. For the WFA $A = (Q_A, \alpha_A, \beta_A, (A_\sigma)_{\sigma\in\Sigma})$ over Σ, and $w = ba$, the configuration $\delta_A(w)$ and the weight $f_A(w)$ are as follows.*

$$\begin{aligned}\delta_A(w) &= \alpha_A^\top A_b A_a \\ &= \begin{pmatrix}1 & 2 & 3\end{pmatrix}\begin{pmatrix}-1 & 1 & 0\\ 0 & 3 & 0\\ -2 & 4 & 0\end{pmatrix}\begin{pmatrix}1 & 2 & -1\\ 3 & 0 & 0\\ 0 & 4 & 0\end{pmatrix} \\ &= \begin{pmatrix}50 & -14 & 7\end{pmatrix}\end{aligned}$$

$$\begin{aligned}f_A(w) &= \alpha_A^\top A_b A_a \beta_A \\ &= \begin{pmatrix}1 & 2 & 3\end{pmatrix}\begin{pmatrix}-1 & 1 & 0\\ 0 & 3 & 0\\ -2 & 4 & 0\end{pmatrix}\begin{pmatrix}1 & 2 & -1\\ 3 & 0 & 0\\ 0 & 4 & 0\end{pmatrix}\begin{pmatrix}0\\ -1\\ 1\end{pmatrix} \\ &= 21\end{aligned}$$

Figure 5.1 illustrates the WFA $A = (Q_A, \alpha_A, \beta_A, (A_\sigma)_{\sigma\in\Sigma})$, where the transitions with weight 0 *are omitted.*

Definition 5.5 (DFA)**.** *A* DFA *is defined much like in Definition 5.1, except that 1) the entries of matrices are* `tt` *and* `ff`*; 2) we replace the use of $+, \times$ with $\vee, \wedge$, respectively; and 3) we impose* determinacy*, that exactly one entry is* `tt` *in each row of A_σ, and that only one entry is* `tt` *in the initial vector α_A.*

The definitions of δ_A and f_A in Definition 5.2–5.3 adapt to DFAs. For $w \in \Sigma^*$, the configuration vector $\delta_A(w) \in \{\texttt{tt}, \texttt{ff}\}^{Q_A}$ has exactly one `tt`*; the state q whose entry is* `tt` *is called the w-*successor *of A. w is* accepted *by A if $f_A(w) = \texttt{tt}$.*

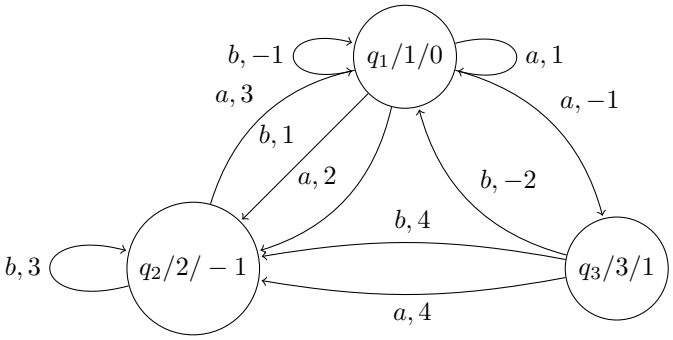


Figure 5.1: An illustration of WFA A in Example 5.4. In a state label "$q/m/n$", q is a state name and m and n are the initial and final values at q's, respectively. In the label "σ, p" of the transition from q_i to q_j, p is $A_\sigma[i,j]$, where $\sigma \in \Sigma$ and $A_\sigma[i,j]$ is the entry of A_σ at row i and column j.

5.2.1 Recurrent Neural Networks

Our view of a recurrent neural network (RNN) is almost a black-box. We need only the following two operations: feeding an input word $w \in \Sigma^*$ and observing its output (a real number); and additionally, observing the internal state (a vector) after feeding a word. This allows us to model RNNs in the following abstract way.

Definition 5.6 (RNN)**.** *Let $d \in \mathbb{N}$ be a natural number called a* dimension*. A (real-valued)* RNN *is a triple $R = (\alpha_R, \beta_R, g_R)$, where $\alpha_R \in \mathbb{R}^d$ is an* initial state*, $\beta_R \colon \mathbb{R}^d \to \mathbb{R}$ is an* output function*, and $g_R \colon \mathbb{R}^d \times \Sigma \to \mathbb{R}^d$ is called a* transition function*. The set $\mathbb{R}^d$ is called a* state space*.*

Definition 5.7 (RNN configuration $\delta_R(w)$, output $f_R(w)$)**.** *Let R be the RNN in Definition 5.6. The transition function g_R naturally extends to words as follows: $g_R^* \colon \mathbb{R}^d \times \Sigma^* \to \mathbb{R}^d$, defined inductively by $g_R^*(x, \varepsilon) = x$ and $g_R^*(x, w\sigma) = g_R\big(g_R^*(x,w), \sigma\big)$, where $w \in \Sigma^*$ and $\sigma \in \Sigma$.*

The configuration $\delta_R(w)$ *of the RNN R at a word w is defined by $\delta_R(w) = g_R^*(\alpha_R, w)$. The* output $f_R(w) \in \mathbb{R}$*, of R for the input w, is defined by $f_R(w) = \beta_R\big(\delta_R(w)\big)$.*

5.2.2 Angluin's L* Algorithm

Angluin's L*-algorithm learns a given DFA B by a series of *membership* and *equivalence* queries. We sketch the algorithm; see [6] for details. Its outline is in Figure 5.2.

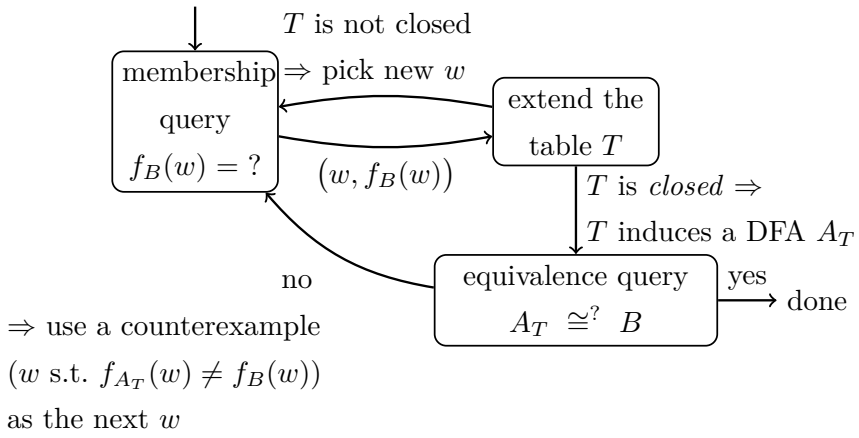


Figure 5.2: An outline of Angluin's L* algorithm. The target DFA is B; a table T gets gradually extended, yielding a DFA A_T when it is closed. See also Figure 5.3a

$\mathcal{A}\backslash\mathcal{T}$	ε	0	1
ε	tt	ff	ff
0	ff	tt	ff
⋮	⋮	⋮	⋮
011	ff	tt	ff

(a) A table T for DFA learning

$\mathcal{A}\backslash\mathcal{T}$	ε	0	1
ε	0.5	0	0.5
0	0	1	0
⋮	⋮	⋮	⋮
001	0.2	0.4	0.4

(b) A table T for WFA learning

Figure 5.3: Observation tables for L*-style algorithms

A *membership query* is a black-box observation of the DFA B: it feeds B with a word $w \in \Sigma^*$; and obtains $f_B(w) \in \{\texttt{tt}, \texttt{ff}\}$, i.e., whether w is accepted by B.

The core of the algorithm is to construct the *observation table* T; see Figure 5.3a. The table has words as the row and column labels; its entries are either tt or ff. The row labels are called *access words*; the column labels are *test words*. We let $\mathcal{A}, \mathcal{T}$ stand for the sets of access and test words. The entry of T at row $u \in \mathcal{A}$ and column $v \in \mathcal{T}$ is given by $f_B(uv)$—a value that we can obtain from a suitable membership query.

Therefore we extend a table T by a series of membership queries. We do so until T becomes closed; this is the top loop in Figure 5.2. A table T is *closed* if, for any access word $u \in \mathcal{A}$ and $\sigma \in \Sigma$, there is an access word $u' \in \mathcal{A}$ such that

$$f_B(u\sigma\, v) = f_B(u'v) \quad \text{for each test word } v \in \mathcal{T}. \tag{5.1}$$

The closedness condition essentially says that the role of the extended word $u\sigma$ is already covered by some existing word $u' \in \mathcal{A}$. The notion of "role" here, formalized in (5.1), is a restriction of the well-known Myhill–Nerode relation, from all words $v \in \Sigma^*$ to $v \in \mathcal{T}$.

A closed table T induces a DFA A_T (Figure 5.2), much like in the Myhill–Nerode theorem. We note that the resulting DFA A_T is necessarily minimized. The DFA A_T undergoes an *equivalence query* that asks if $A_T \cong B$; an equivalence query is answered with a counterexample—i.e., $w \in \Sigma^*$ such that $f_{A_T}(w) \neq f_B(w)$—if $A_T \ncong B$.

The L* algorithm is a *deterministic* learning algorithm (at least in its original form), unlike many recent learning algorithms that are statistical. The greatest challenge in practical use of the L* algorithm is to answer equivalence queries. When B is a finite automaton that generates a regular language, there is a complete algorithm for deciding the language equivalence $A_T \cong B$. However, if we use a more expressive model in place of a DFA B, checking $A_T \cong B$ becomes a nontrivial task.

5.2.3 L* Algorithm for WFA Learning

The classic L* algorithm for learning DFAs has seen a *weighted* extension [12]: it learns a WFA B, again via a series of membership and equivalence queries. The overall structure of the WFA learning algorithm stays the same as in Figure 5.2; here we highlight major differences.

Firstly, the entries of an observation table T are now real numbers, reflecting the fact that the value $f_B(uv)$ for a WFA B is in $\mathbb{R}$ instead of in $\{\mathtt{tt}, \mathtt{ff}\}$ (see Definition 5.3). An example of an observation table is given in Figure 5.3b.

Secondly, the notion of closedness is adapted to the weighted (i.e., linear-algebraic) setting, as follows. A table T is *closed* if, for any access word $u \in \mathcal{A}$ and $\sigma \in \Sigma$, the vector $\big(f_B(u\sigma\, v)\big)_{v\in\mathcal{T}} \in \mathbb{R}^{|T|}$ can be expressed as a linear combination of the vectors in $\big\{\big(f_B(u'v)\big)_{v\in\mathcal{T}} \,\big|\, u' \in \mathcal{A}\big\}$. Note that the vector $\big(f_B(u'v)\big)^{\top}_{v\in\mathcal{T}}$ in the latter set is precisely the row vector in T at row u'.

For example, the table T in Figure 5.3b is obviously closed, since the three row vectors are linearly independent and thus span the whole $\mathbb{R}^3$. The above definition of closedness comes natural in view of Definition 5.2. For a WFA, a configuration (during its execution) is not a single state, but a *weighted superposition* $x \in \mathbb{R}^{Q_A}$ of states. The closedness condition asserts that the role of $u\sigma$ is covered by a suitable superposition of words $u' \in \mathcal{A}$. The construction of the WFA A_T from a closed table T (see Figure 5.2) reflects this intuition. See [12]. We note that the resulting A_T is minimal, much like in Section 5.2.2.

In the literature [12], an observation table T is presented as a so-called *Hankel* matrix. This opens the way to further extensions of the method, such as an approximate learning algorithm via the singular-value decomposition (SVD).

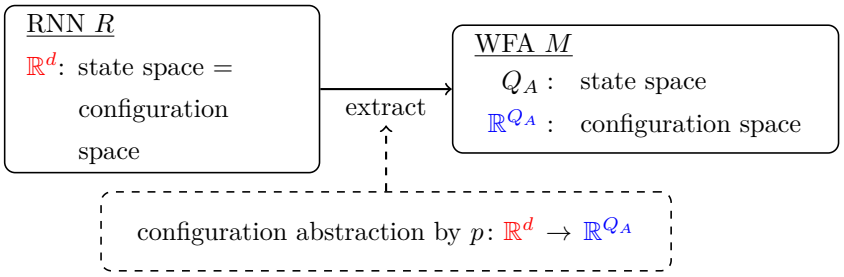


Figure 5.4: An outline of our WFA extraction

5.3 WFA Extraction from an RNN

We present our main contribution, namely a procedure that extracts a WFA from a given RNN. After briefly describing its outline (that is the same as Figure 5.2), we focus on the greatest challenge of answering equivalence queries.

5.3.1 Procedure Outline

Our procedure uses the weighted L* algorithm sketched in Section 5.2.3. As we discussed in Section 5.2.2, the greatest challenge is how to answer equivalence queries; our novel approach is to use *regression* and synthesize what we call a *configuration abstraction function* $p\colon \mathbb{R}^d \to \mathbb{R}^{Q_A}$. See Figure 5.4.

The outline of our procedure thus stays the same as in Figure 5.2, but we need to take care about noisy outputs from an RNN because they prevent the observation table from being precisely closed (in the sense of Section 5.2.3). To resolve this issue, we use a noise-tolerant algorithm [12] which approximately determines whether the observation table is closed. This approximate algorithm employs SVD and cuts off singular values that are smaller than a threshold called *rank tolerance*. In the choice of a rank tolerance, we face the trade off between accuracy and regularity. A small rank tolerance results in accurate learning basically but can cause overfitting for short words and huge error for long words. A large rank tolerance results in rough learning but prevents such overfitting. To balance the rank tolerance, we start from a big initial rank tolerance τ, and if it is too big then we decay it by multiplying by $r (0 < r < 1)$. We know that the rank tolerance is too big if the equivalence query returns the same counterexample twice because it means the counterexample was ignored. Overall, we obtain the WFA Extraction procedure (Algorithm 4).

Algorithm 4 WFA Extraction

1: **Input:** Target RNN R, initial rank tolerance τ, decay rate of rank tolerance $r (0 < r < 1)$
2: `result` $\leftarrow$ `None`
3: `result`$'$ $\leftarrow$ `None`
4: `observed` $\leftarrow \{ (\varepsilon, f_R(\varepsilon)) \}$
5: Construct a WFA A by `observed`
6: **loop**
7: `result`$'$ $\leftarrow$ (result of equivalence query for A)
8: **insert** $(result', f_R(result'))$ **to** `observed`
9: **if** `result` $=$ `result`$'$ **then** ▷ The latest counterexample did not work.
10: $\tau \leftarrow r\tau$ ▷ Decrease the rank tolerance τ
11: **else**
12: Update A by $(\texttt{result}', f_R(\texttt{result}'))$ with τ ▷ τ is the current rank tolerance.
13: **if** `result` $=$ `Equivalent` **then**
14: **break**
15: `result` $\leftarrow$ `result`$'$
16: **return** A

5.3.2 Equivalence Queries for WFAs and RNNs

Algorithm 5 shows our procedure to answer an equivalence query. The procedure Ans-EqQ is the main procedure, and it returns either `Equivalent` or a counterexample word (as in Figure 5.2). It calls the auxiliary procedure Consistent?, which decides whether we refine the current configuration abstraction function $p\colon \mathbb{R}^d \to \mathbb{R}^{Q_A}$ in Figure 5.4 (Line 10).

5.3.2.1 Best-First Search for a Counterexample

The procedure Ans-EqQ is essentially a best-first search for a *counterexample*, that is, a word $h \in \Sigma^*$ such that the difference of the output values $f_R(h)$ from the RNN and $f_A(h)$ from the WFA is larger than *error tolerance* $e (> 0)$. We first outline Ans-EqQ and then go into the technical detail.

We manage counterexample candidates by the priority queue `queue`, which gives a higher priority to a candidate more likely to be a counterexample. The already investigated words are in the set `visited`. The queue `queue` initially contains only the empty word ε (Line 3).

We search for a counterexample in the main loop starting from Line 4. Let h be a word popped from `queue`, that is, the candidate most likely to be a proper counterexample among

Algorithm 5 Answering equivalence queries

1: **procedure** ANS-EQQ

Input: RNN $R = (\alpha_R, \beta_R, g_R)$, WFA $A = (Q_A, \alpha_A, \beta_A, (A_\sigma)_{\sigma\in\Sigma})$, error tolerance $e > 0$ and concentration threshold $M \in \mathbb{N}$

Output: a counterexample, or `Equivalent`

2: Initialize $p\colon \mathbb{R}^d \to \mathbb{R}^{Q_A}$ so that $p(\delta_R(\varepsilon)) = \delta_A(\varepsilon)$

3: `queue` $\leftarrow \langle\varepsilon\rangle$; `visited` $\leftarrow \emptyset$

4: **while** `queue` is non-empty **do**

5: $h \leftarrow$ **pop**(`queue`) ▷ Pop the element of the maximum priority

6: **if** $|f_R(h) - f_A(h)| \geq e$ **then**

7: **return** h ▷ return a counterexample

8: `result` $\leftarrow$ CONSISTENT?$(h,$ `visited`$, p)$

9: **if** `result` $=$ `NG` **then**

10: learn p by regression, so that $p(\delta_R(h')) = \delta_A(h')$ holds for all $h' \in$ `visited` $\cup \{h\}$

11: `visited` $\leftarrow$ `visited` $\cup \{ h \}$

12: `visited`$' \leftarrow p(\delta_R($`visited`$))$

13: $\#vn \leftarrow |\{ x \in$ `visited`$' \mid x \simeq_A p(\delta_R(h)) \}|$

14: **if** $\#vn \leq M$ **then**

15: $\mathsf{pr} \leftarrow \min_{h'\in\texttt{visited}\setminus\{h\}} d(p(\delta_R(h)), p(\delta_R(h')))$ ▷ d is the Euclidean distance

16: **push** $h\sigma$ **to** `queue` with priority pr **for** $\sigma \in \Sigma$

17: **return** `Equivalent`

the words in the queue. If h is a counterexample, ANS-EQQ returns it (Lines 6–7). Otherwise, after refining the configuration abstraction function p (Lines 8–10), new candidates $h\sigma$, the extension of h with character σ, are pushed to `queue` with their priorities *only if* the neighborhood of h in the state space of the WFA A does not contain sufficiently many already investigated words—i.e., only if it has not been investigated sufficiently (Lines 13–16). This is because, if the neighborhood of h has been investigated sufficiently, we expect that the neighborhoods of the new candidates $h\sigma$ have also been investigated and, therefore, that the words $h\sigma$ do not have to be investigated furthermore. We use p: 1) to decide if the neighborhood of h has been investigated sufficiently (Line 13); and 2) to calculate the priorities of the new candidates (Line 15). Note that we add h to `visited` in Line 11 since it has been investigated there. If all the candidates are not a counterexample, ANS-EQQ returns `Equivalent` (Line 17).

5.3.2.2 Configuration Abstraction Function p

To use p for the aformationed purposes, the property we expect from the configuration abstraction function $p\colon \mathbb{R}^d \to \mathbb{R}^{Q_A}$ is as follows:

$$p(\delta_R(h)) \approx \delta_A(h) \quad \text{for as many } h \in \Sigma^* \text{ as possible.} \tag{5.2}$$

See Definition 5.2 and 5.7 for $\delta_A\colon \Sigma^* \to \mathbb{R}^{Q_A}$ and $\delta_R\colon \Sigma^* \to \mathbb{R}^d$, respectively. To synthesize such a function p, we employ *regression* using the data $\left\{ \left(\delta_R(h'), \delta_A(h')\right) \in \mathbb{R}^d \times \mathbb{R}^{Q_A} \,\middle|\, h \in \texttt{visited} \right\}$. See Line 10. Note that we can use any regression method to learn p.

We refine p during the best-first counterexample search. Specifically, in Line 8, we use the procedure Consistent? to check if the current p—obtained by regression—is consistent with a counterexample candidate h. The consistency means that $p(\delta_R(h))$ and $\delta_A(h)$ are close to each other, which is formalized by the relation $\simeq_A$ defined later. If the check fails (i.e., if Consistent? returns `NG`), we refine p by regression to make p consistent with h (and the already investigated words in `visited`). See Line 10.

5.3.2.3 Consistency Checking by Consistent?

The procedure Consistent? in Line 8 is defined as follows: it returns `NG` if there exists $h' \in \texttt{visited}$ such that

$$\delta_A(h') \not\simeq_A p(\delta_R(h')) \text{ and } p(\delta_R(h')) \simeq_A p(\delta_R(h)), \tag{5.3}$$

and returns `OK` otherwise. The basic idea of Consistent? is to return `NG` if $p(\delta_R(h)) \not\simeq_A \delta_A(h)$ because it means the violation of the desired property (5.2). However, to reduce the run-time cost of refining p and to prevent learning from outliers, we adopt the alternative approach presented above, which is taken from [134].

The existence of h' satisfying the condition (5.3) approximates the violation of the property (5.2) in the following sense. If there is a word h' satisfying the first part of the condition (5.3), p has to be refined because we find the property (5.2) violated with h'. The second part of the condition (5.3) means that h' seems to behave similarly to h according to the configuration abstraction function p. We expect the second part to prevent p from being refined with outliers because the neighborhoods of the words used for refining p must have been investigated twice or more.

5.3.2.4 Equivalence Relation $\simeq_A$

For a given WFA A, we define the relation $\simeq_A$ in the configuration space $\mathbb{R}^{Q_A}$ by

$$\vec{x} \simeq_A \vec{y} \iff \sum_{i=1}^{|Q_A|} \beta_i^2 (x_i - y_i)^2 < \frac{e^2}{|Q_A|}, \tag{5.4}$$

where β is the final vector of the WFA A. It satisfies the following.

1. If $x \simeq_A y$ holds, the difference of the output values for the configurations x and y of the WFA A is smaller than the error tolerance e, that is, $|(x - y) \cdot \beta| < e$.
2. If the i-th element of the final vector β becomes large, the neighborhood $\{ y \in \mathbb{R}^{Q_A} \mid x \simeq_A y \}$ of x shrinks in the direction of the i-th axis.
3. The neighborhood defined by $\simeq_A$ is an ellipsoid—a reasonable variant of an open ball as a neighborhood.

5.3.2.5 A Heuristic for Equivalence Checking of a WFA and an RNN

Although the best-first search above works well, we introduce an additional heuristic to improve the run-time performance of our algorithm furthermore. The heuristic deems R and A to be equivalent if word h previously popped from `queue` is so long that it is impossible to occur in the training of the RNN. This heuristic is based on the expectation that, when an impossible word is the most likely to be a counterexample, all possible words are unlikely to be a counterexample, and so R and A are considered to be equivalent. This heuristic is adopted immediately after popping h (Line 5), as follows: we suppose that the maximum length L of possible words is given; and, if the length of h is larger than L, Ans-EqQ returns `Equivalent`. We confirm the usefulness of this heuristic in Section 5.4 empirically.

5.3.2.6 Termination of the Procedure

Algorithm 5 does not always terminate in a finite amount of time. If the procedure does not find any counterexample at Line 7 and the points $p(\delta_R(\texttt{visited}))$ are so scattered in the configuration space that the value $\#vn$ at Line 13 is always small, words are always pushed to `queue` at Line 16. In that case, the condition to exit the main loop at Line 4 is never satisfied.

5.3.3 Comparison with Weiss et al., 2018

Our WFA extraction method can be seen as a weighted extension of the L*-based procedure [134] to extract a DFA from an RNN. Note that a WFA defines a function of type $\Sigma^* \to \mathbb{R}$ and a DFA defines a function of type $\Sigma^* \to \{ \texttt{tt}, \texttt{ff} \}$.

The main technical novelty of the method in [134] is how to answer equivalence queries. It features the *clustering* of the state space $\mathbb{R}^d$ of an RNN into finitely many clusters, using support-vector machines (SVMs). Each cluster of $\mathbb{R}^d$ is associated with a state $q \in Q_A$ of the DFA.

Our theoretical observation is that such clustering amounts to giving a function $p\colon \mathbb{R}^d \to Q_A$. Moreover, for a DFA, Q_A is the configuration space (as well as the state space, see Definition 5.5). Therefore, our WFA extraction method can be seen as an extension of the DFA extraction procedure in [134].

5.4 Experiments

We conducted experiments to evaluate the utility of our regression-based WFA extraction method. Specifically, we pose the following questions.

RQ1 Do the WFAs extracted by our algorithm approximate the original RNNs accurately?

RQ2 Does our algorithm work well with RNNs whose expressivity is beyond WFAs?

RQ3 Do the extracted WFAs run more efficiently, specifically in inference time, than given RNNs?

For **RQ1**, we compared the performance with a baseline algorithm (a straightforward adaptation of [12]'s L*algorithm). Here we focused on "automata-like" RNNs, that is, those RNNs trained from an original WFA $A^\bullet$. For **RQ2**, we used an RNN that exhibits "context-free" behaviors.

Experimental Setting We implemented our method in Python. We write $\mathbf{RGR}(n)$ for the algorithm where the concentration threshold M in Ans-EqQ is set to n; other parameters are fixed as follows: error tolerance $e = 0.05$ and heuristic parameter $L = 20$. We adopt the Gaussian process regression (GPR) provided by scikit-learn as a configuration abstraction function p (we also tried the kernel ridge regression but GPR worked empirically better). Throughout the experiments, our RNNs are 2-layer LSTM networks with dimension size 50, implemented by TensorFlow. The experiments were on a g3s.xlarge instance on Amazon Web Service (May 2019), with a NVIDIA Tesla M60 GPU, 4 vCPUs of Xeon E5-2686 v4 (Broadwell), and 8GiB memory.

5.4.1 RQ1: Extraction from RNNs Modeling WFAs

This experiment examines how well our algorithm work for RNNs modeling WFAs. To do so, we first train RNNs using randomly generated WFAs; we call those WFAs the *origins*.

Table 5.1: Experiment results, where we extracted a WFA A from an RNN R that is trained to mimic the original WFA $A^\bullet$. In each cell "n/m", "n" denotes the average of MSEs between A and R (the unit is 10^{-4}), taken over five random WFAs $A^\bullet$ of the designated alphabet size $|\Sigma|$ and the state-space size $|Q_{A^\bullet}|$. "m" denotes the average running time (the unit is second). The "Total" row describes the average over all the experiment settings. The highlighted cell designates the best performer in terms of errors. Timeout was set at 10,000 sec. **RGR**(2–5) are our regression-based methods; **BFS**(500–5000) are the baseline. **BFS**(5000) is added to compare the accuracy when the running time is much longer.

□ Results for "more WFA-like" RNNs R, i.e., those RNNs R which are trained from WFAs $A^\bullet$ using a uniformly sampled data set $T \subseteq \Sigma^*$

$(\|\Sigma\|, \|Q_{A^\bullet}\|)$	**RGR**(2)	**RGR**(5)	**BFS**(500)	**BFS**(1000)	**BFS**(2000)	**BFS**(3000)	**BFS**(5000)
$(4, 10)$	2.17 / 286	2.39 / 338	26.8 / 165	9.77 / 279	4.36 / 545	4.07 / 716	2.33 / 1390
$(6, 10)$	2.45 / 1787	2.54 / 1302	6.99 / 386	4.48 / 641	4.08 / 1218	3.15 / 1410	2.28 / 2480
$(10, 10)$	4.68 / 7462	4.46 / 5311	22.5 / 928	11.9 / 1562	5.90 / 3521	4.55 / 3638	3.55 / 5571
$(10, 15)$	5.62 / 8941	5.78 / 8564	21.2 / 2155	10.6 / 4750	7.87 / 5692	5.71 / 7344	5.27 / 7612
$(10, 20)$	3.70 / 7610	3.79 / 7799	6.24 / 2465	10.1 / 2188	6.13 / 3106	3.70 / 5729	3.63 / 7473
$(15, 10)$	7.34 / 9569	5.52 / 10000	13.5 / 3227	8.01 / 6765	6.07 / 7916	5.98 / 8911	6.17 / 8979
$(15, 15)$	8.44 / 10000	5.58 / 9981	16.3 / 2675	9.24 / 4850	7.28 / 5135	9.88 / 7204	6.44 / 8425
$(15, 20)$	9.16 / 7344	5.15 / 7857	13.7 / 2224	7.26 / 3823	6.60 / 5744	4.96 / 5674	4.01 / 9464
Total	5.45 / 6625	4.40 / 6394	15.9 / 1778	8.92 / 3107	6.04 / 4110	5.25 / 5078	4.21 / 6549

□ Results for "realistic" RNNs R with *nonuniform input domains*, i.e., those R which are trained from WFAs $A^\bullet$ using a data set $T \subseteq \Sigma^*$ in which some specific patterns are prohibited

$(\|\Sigma\|, \|Q_{A^\bullet}\|)$	**RGR**(2)	**RGR**(5)	**BFS**(500)	**BFS**(1000)	**BFS**(2000)	**BFS**(3000)	**BFS**(5000)
$(4, 10)$	7.73 / 696	7.07 / 1135	15.0 / 199	7.96 / 424	6.62 / 650	6.61 / 762	9.06 / 1693
$(6, 10)$	4.92 / 1442	7.43 / 1247	1.46 / 552	6.95 / 660	5.90 / 1217	8.78 / 1557	3.54 / 2237
$(10, 10)$	5.02 / 5536	4.28 / 5951	7.70 / 1117	11.0 / 1738	4.77 / 2635	3.52 / 3926	4.52 / 4777
$(10, 15)$	7.15 / 6977	4.35 / 8315	19.4 / 1552	13.8 / 3271	16.8 / 3209	8.57 / 5293	5.08 / 6522
$(10, 20)$	6.98 / 4697	8.06 / 6704	18.6 / 1465	11.8 / 2046	12.7 / 2851	9.03 / 4259	8.01 / 4856
$(15, 10)$	5.97 / 8747	6.77 / 8882	23.3 / 2359	11.2 / 4668	9.88 / 6186	6.24 / 7557	6.02 / 8245
$(15, 15)$	5.78 / 8325	8.71 / 7546	16.6 / 2874	7.31 / 4380	9.92 / 6015	9.89 / 7110	6.40 / 8358
$(15, 20)$	4.60 / 7652	8.56 / 8334	36.9 / 1893	23.7 / 3069	12.8 / 3987	12.0 / 5262	8.38 / 6441
Total	6.02 / 5510	6.90 / 6015	19.0 / 1502	11.7 / 2532	9.92 / 3344	8.08 / 4466	6.38 / 5391

Then, we evaluate our algorithm compared with a baseline from two points: accuracy of the extracted WFAs against the trained RNNs and running times of the algorithms. We report the results after presenting the details of the baseline, how to train RNNs, and how to evaluate the two algorithms.

The Baseline Algorithm: BFS(n) As our extraction algorithm, the baseline algorithm **BFS**(n), which is parameterized over an integer n, is a straightforward adaptation of [12]'s L*algorithm. The difference is that equivalence queries in the baseline are implemented in

breath-first search, as follows. Let R be a given RNN and A be a WFA being constructed. For each equivalence query, the baseline searches for a word w such that $|f_A(w) - f_R(w)| > e$ (where $e = 0.05$), in the breadth-first manner. If such a word w is found it is returned as a counterexample. The search is restricted to the first $i + n$ words, where i is the index of the counterexample word found in the previous equivalence query. If no counterexample w is found within this search space, the baseline algorithm deems A to be equivalent to R. Obviously, if n is larger, more counterexample candidates are investigated.

Target RNNs R trained from WFAs $A^\bullet$ Table 5.1 reports the accuracy of the extracted WFAs A, where the target RNNs R are obtained from original WFAs $A^\bullet$ in the following manner. Given an alphabet Σ of a designated size (the leftmost column), we first generated a WFA $A^\bullet$ such that 1) the state-space size $|Q_{A^\bullet}|$ is as designated (the leftmost column), and 2) the initial vector $\alpha^\bullet_A$, the final vector $\beta^\bullet_A$, and the transition matrix $A^\bullet_\sigma$ are randomly chosen (with normalization so that its outputs are in $[0, 1]$). Then we constructed a dataset T by sampling 9000 words w such that $w \in \Sigma^*$ and $|w| \leq 20$; this T was used to train an RNN R, on the set of input-output pairs of $w \in T$ and $f_{A^\bullet}(w)$ for 10 epochs.

A simple way to sample words $w \in T$ is by the uniform distribution. With T covering the input space of the WFA $A^\bullet$ uniformly, we expect the resulting RNN R to inherit properties of $A^\bullet$ well. The top table in Table 5.1 reports the results in this "more WFA-like" setting.

However, in many applications, the input domain of the data used for training RNNs are nonuniform, sometimes even excluding some specific patterns. To evaluate our method in such realistic settings, we conducted another set of experiments whose results are in the bottom of Table 5.1. Specifically, for training R from $A^\bullet$, we used a dataset T that only contains those words $\sigma_1\sigma_2 \ldots \sigma_n$ which satisfy the following condition: if $\sigma_i = \sigma_j$ $(i < j)$, then $\sigma_i = \sigma_{i+1} = \ldots = \sigma_{j-1} = \sigma_j$. For example, for $\Sigma = \{a, b, c\}$, *aabccc* and *baaccc* may be in T, but *aaba* may not.

Evaluation In order to evaluate accuracy, we calculated the mean square error (MSE) of the extracted WFA A against the RNN R, using a dataset V of words sampled from an appropriate distribution, namely the one used in training the RNN R from $A^\bullet$. The dataset V is sampled so that it does not overlap with the training dataset T for R.

Results and Discussions In the experiments in Table 5.1, we considered 8 configurations for generating the original WFA $A^\bullet$ (the leftmost column). The unit of MSEs are -4—given also that the outputs of the original WFAs $A^\bullet$ are normalized to $[0, 1]$, we can say that the MSEs are small enough.

In the top table in Table 5.1 (the "more WFA-like" setting), **BFS**(5000) and **RGR**(5) achieved the first- and second-best performance in terms of accuracy, respectively (see the "Total" row). More generally, we can find the trend that, as an extraction runs longer, it

performs better. We conjecture its reason as follows. Recall that all the RNNs are trained on words sampled from the uniform distribution. This means that all words would be somewhat informative to approximate the RNNs. As a result, the performance is more influenced by the amount of counterexamples—i.e., how long time extraction takes—than on their "qualities."

The exception of this trend is $\mathbf{RGR}(2)$, which took a longer time but performed worse than $\mathbf{BFS}(3000)$, $\mathbf{BFS}(5000)$, and $\mathbf{RGR}(5)$. In particular, $\mathbf{RGR}(2)$ performed well for smaller alphabets ($|\Sigma| \in \{4, 6, 10\}$) but not so when $|\Sigma| = 15$. The role of the parameter M in $\mathbf{RGR}(M)$ (i.e., in Algorithm 5) is a threshold to control how many words configuration regions of a WFA are investigated with. Thus, we conjecture that the use of too small M limits the input space to be investigated excessively, which is more critical as the input space gets larger, eventually biasing the counterexamples h (in Algorithm 5), though the RNNs are trained on the uniform distribution, and making refinement of WFAs less effective.

In the bottom table in Table 5.1 (the "realistic" setting), $\mathbf{RGR}(2)$ performs significantly better than the other (and the best among all the procedures) in terms of accuracy. This is the case even for a large alphabet ($|\Sigma| = 15$). This indicates that, in the cases that an RNN is trained with a nonuniform dataset, making the investigated input space larger with a big M could even degrade the accuracy performance. A possible reason for this degradation is as follows. Some words (such as aba) are prohibited in the sample set T, and the behaviors of the RNN R for those prohibited words are unexpected. Therefore, those prohibited words should not be useful to refine a WFA. The use of small M could prevent such meaningless (or even harmful) counterexamples h from being investigated. This discussion raises another question: how can we find an optimal M? We leave it as a future work.

Let us briefly discuss the sizes of the extracted WFAs. The general trend is that the extracted WFAs A have a few times greater number of states than the original WFAs $A^\bullet$ used in training R. For example, in the setting of the top table in Table 5.1, for $|\Sigma| = 15$ and $|Q_{A^\bullet}| = 20$, the average number of the states of the extracted A was 38.2.

5.4.2 RQ2: Expressivity beyond WFAs

We conducted experiments to examine how well our method works for RNNs modeling languages that cannot be expressed by any WFA. Specifically, we used an RNN that models the following function $\texttt{wparen}\colon \Sigma^* \to [0, 1]$: $\Sigma = \{(,), 0, 1, ..., 9\}$, $\texttt{wparen}(w) = 1 - (1/2)^N$ if all the parentheses in w are balanced (here N is the depth of the deepest balanced parentheses in w); and $\texttt{wparen}(w) = 0$ otherwise. This `wparen` is a weighted variant of a (non-regular) language of balanced parentheses. For instance, $\texttt{wparen}(\text{“((3)(7)))”}) = 0$, $\texttt{wparen}(\text{“((3)(7))”}) = 1 - (1/2)^2 = 3/4$, and $\texttt{wparen}(\text{“(a)(b)(c)”}) = 1/2$.

We trained an RNN R as follows. We generated datasets T_{good} and T_{bad}, and trained an RNN R on the set of input-output pairs of $w \in T_{\text{good}} \cup T_{\text{bad}}$ and $\texttt{wparen}(w)$. The dataset T_{good} consists of randomly generated words where all the parentheses are balanced; T_{bad} is constructed similarly, except that we apply suitable mutation to each word, which most likely makes the parentheses unbalanced. The detailed procedure is as follows:

1. We make 5000 words of random balanced parentheses made only of $\{\,(,)\,\}$. There is a one-to-one correspondence between words of balanced parentheses of length $2n$ and paths from the bottom-left to the top-right in the grid of size $n \times n$ whose bottom-right half is removed, so we can obtain such random words by generating the paths randomly and converting them into the words. For example, "(())" or "(()())" can be made.
2. We insert random characters in $\{\,0, 1, \ldots, 9\,\}$ into the words generated in Step 1. This generates 5000 words of random balanced words made of $\{\,(,), 0, 1, \ldots, 9\,\}$. For example, "(0(1))" or "((12340()))" can be made.
3. We run the same procedure as Step 1 and obtain 5000 words of random balanced parentheses.
4. We mutate the words in Step 3 and make them into 5000 random unbalanced parentheses made only of $\{\,(,)\,\}$. The mutation rules are as follows: 1) duplicate a random character; 2) delete a random character; and 3) exchange a random pair of adjacent characters. These rules are repeatedly applied—each time throwing a fair coin—until we get the head of the coin. Note that the mutation can make a balanced word into another balanced word. For example, "(()", "((((", or "()" can be made (only the last one is balanced).
5. We insert random characters in $\{\,0, 1, \ldots, 9\,\}$ into the words generated in Step 4. This generates 5000 words of random unbalanced words made of $\{\,(,), 0, 1, \ldots, 9\,\}$.
6. We combine the result of Step 2 and 5 and get 10000 words. Almost the half of the words are balanced and the other half are unbalanced. We pick 9000 random words from the words and use them as the training data; the remaining 1000 are used as the test data.

Figure 5.5 shows the WFAs extracted from R. Remarkable observations here are as follows.

- The shapes of the WFAs—obtained by ignoring clearly negligible weights—give rise to NFAs that recognize balanced parentheses up-to a certain depth.
- As the parameter M in $\mathbf{RGR}(M)$ grows, the recognizable depth bound grows: depth one with $\mathbf{RGR}(5)$; and depth two with $\mathbf{RGR}(15)$.

We believe these observations demonstrate important features, as well as limitations, of our

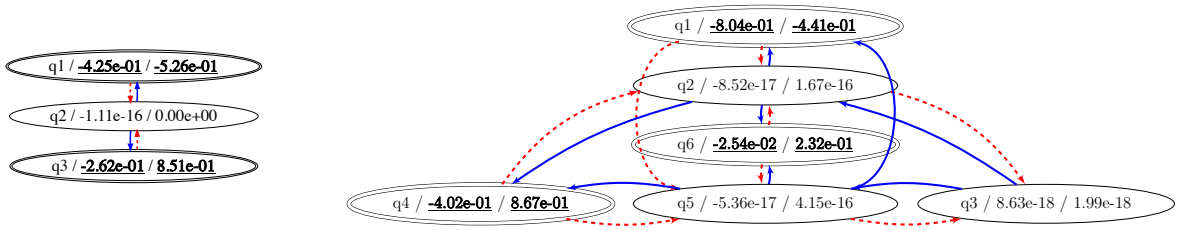


Figure 5.5: The WFAs extracted by **RGR**(5) (left) and **RGR**(15) (right). In a state label "$q/m/n$", q is the state name, m is the initial value and n is the final value. Bigger values are underlined; other values are negligibly small. The dotted and solid edges are labelled with "`(`" and "`)`" respectively; the edges with labels $0, 1, \ldots, 9$ are omitted. The edge weights are omitted for simplicity too; the weight threshold for showing transitions is 0.01. Full details on these WFAs are in Appendix A.1.

method. Overall, the extracted WFAs expose interpretable structures hidden in an RNN: the NFA structures in Figure 5.5 are human-interpretable (they are easily seen to encode bounded balancedness) and machine-processable (such as determinization and minimization). It is also suggested that the parameter M gives us flexibility in the trade-off of extraction cost and accuracy. At the same time, we can only obtain a truncated and regularized version of the RNN structure—this is an inevitable limitation as long as we use the formalism of WFAs.

We also note that, in each of the two extracted WFAs, the transition matrices A_σ are similar for all $\sigma \in \{0, 1, \ldots, 9\}$ (the entries at the same position have the same order). This is as expected too, since the function `wparen` does not distinguish the characters $0, 1, \ldots, 9$.

5.4.3 RQ3: Accelerating Inference Time

We conducted experiments about inference times, comparing the original RNNs R with the WFAs A that we extracted from R. We used the same RNNs R and WFAs A as in Section 5.4.1, where the latter are extracted using **RGR**(2–5) and **BFS**(500–5000). We note that the inference of RNNs utilizes GPUs while that of WFAs is solely done by CPUs.

On average, the inference time of the target RNNs was 29.975 milliseconds, while that of the extracted WFAs was 0.023 milliseconds. Therefore, on average, the inference of the extracted WFAs was about 1300 times faster than that of the target RNNs.

This demonstrates the potential use of the extracted WFAs as a computationally cheaper surrogate for RNNs. We attribute the acceleration to the following: 1) WFAs use only linear computation while RNNs involve nonlinear ones; and 2) overall, extracted WFAs are smaller in size. Provided that the accuracy of extracted WFAs can be high (as we observed in Section

5.4.1), we believe the replacement of RNNs by WFAs is a viable option in some application scenarios.

5.5 Conclusions

We proposed a method that extracts a WFA from an RNN, focusing on RNNs that take a word $w \in \Sigma^*$ and return a real value. We used regression to investigate and abstract the internal states of RNNs. We experimentally evaluated our method, comparing its performance with a baseline whose equivalence queries are based on simple breadth-first search.

5.6 Future Work

One future work is a detailed comparison with other methods for model compression. Another future work is to use machine learning methods to find a counterexample in the equivalence query, such as reinforcement learning [94] adversarial attacks [105], and acquisition functions of GPR. Finally, we need a means to optimize the parameter M of our method for a specific problem. It may also be helpful to extend our method so that the investigated words can be restricted to a fixed language $L \subset \Sigma^*$. If L identifies the input space of the training dataset for RNNs, we could avoid investigating the input space on which the RNNs are not trained, and therefore we could seek only "meaningful" counterexamples even when using large M.

Chapter 6

Learning Weighted Finite Automata over the Max-Plus Semiring, and Beyond

This chapter is based on joint work with Masaki Waga, Taro Sekiyama, and Ichiro Hasuo.

6.1 Introduction

6.1.1 Background

The topic of *active automata learning* has seen years of extensive study. The initial success is brought by Angluin's L^* *algorithm* [6], which repeatedly issues so-called *membership queries*, organize their answers in a table, and use the table to construct a deterministic finite automaton (DFA). A black-box system can be thought of as an oracle that answers membership queries. This way, the algorithm constructs a DFA that serves as a white-box surrogate of the black-box system.

The mathematical cleanness of the L^* algorithm has led to a number of extensions. A notable one is a quantitative *weighted* extension [12], where membership queries are answered by real numbers instead of Boolean values (accept or reject). The algorithm is best understood in linear algebra terms, where the notion of table is alternatively described by (a finite subblock of) the so-called *Hankel matrix*. The output of the algorithm is a *weighted finite automaton* (WFA), where the notions of initial state, final state, and transition are all weighted by real numbers.

WFA is a general notion that is parametric in the choice of an underlying *semiring*. A semiring $\mathbb{S}$ is an algebraic structure with operations $\oplus$ and $\otimes$, subject to certain equation axioms. In a WFA over a semiring $\mathbb{S}$, weights are taken from its underlying set; they are aggregated along a path (i.e. a sequence of transitions) by $\otimes$; and these path weights are

"summed up" by $\oplus$ over different paths.

The aforementioned work [12] is over the *real semiring*, which is carried by the set of real numbers and whose semiring operations $\oplus$ and $\otimes$ are the usual addition ($+$) and multiplication ($\times$) of real numbers. Another well-known example of a semiring is the *max-plus semiring*, a member of the family called *tropical semirings*, where $\oplus$ takes the maximum of real numbers and $\otimes$ is the usual addition $+$. WFAs over tropical semirings have applications in multiple domains, from computer science [3, 24] to control theory [41, 42]. One viable intuition is that weights are rewards: they are accumulated along a path; and a strategy chooses the path that gives the maximum reward.

6.1.2 Active WFA Learning over General Semirings

In this chapter, we study active WFA learning over the max-plus semiring. L*-style WFA learning over general semirings is recently studied in [129]. We found that the algorithm used there—a natural adaptation of the algorithm in [12]—has a "consistency" issue, one that makes the tables bigger than necessary, increases the number of queries and thus potentially harms the performance. We present a theoretical fix—by a mathematically clean notion of *column-closedness*—and an algorithm that implements the fix.

6.1.3 Outline of L*-Style Algorithms

We first illustrate the issue in the max-plus WFA learning that we identified. It arises from the notion of *consistency* of L* tables. Towards its explanation, we first give an outline of L*-style algorithms.

The problem of active WFA learning in the style of L* is formulated as follows. Here $\mathbb{S}$ is the underlying semiring. Some notions here will be formally introduced later.

Problem 1 (active WFA learning) Let $f\colon \Sigma^* \to \mathbb{S}$ be a function (called a *weighted language*); it is hidden from us. We are given the following two oracles.

- A *membership oracle* m that takes a word $w \in \Sigma^*$ and returns $f(w) \in \mathbb{S}$.
- An *equivalence oracle* e that takes a WFA A' over $\mathbb{S}$, and
  - returns the symbol `Eq.` if the weighted language $f_{A'}$ induced by the WFA A' coincides with f; and
  - returns w otherwise, where w is a word such that $f_{A'}(w) \neq f(w)$. This w is called a *counterexample*.

The desired output is a WFA A' that mimics f (i.e. $f_{A'}(w) = f(w)$ for any word $w \in \Sigma^*$,

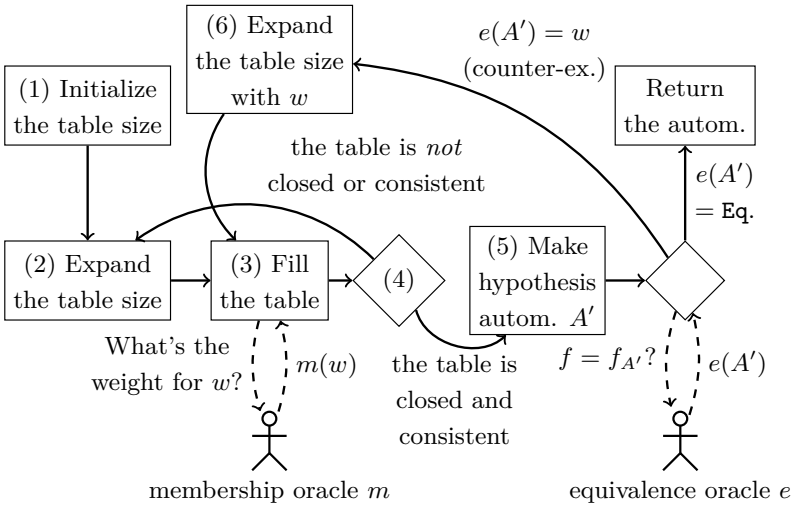


Figure 6.1: L*-style algorithms, an outline. Dashed lines indicate interaction with the oracles. Algorithms differ in what exactly they require in "closedness" and "consistency."

Table 6.1: A Hankel submatrix in the setting of real-number weights

$P \backslash S$	ϵ	a
ϵ	1.3	2.6
a	2.6	3.4
ab	3.5	4.0
b	2.8	3.0

using the notation introduced later).

An L*-style algorithm solves the problem in the following way (Figure 6.1). It issues multiple membership queries, recording the oracle's answers in a table. An example of a table is shown in Table 6.1 (for real-number weights), where rows and columns are indexed by words $p \in P$ and $s \in S$ (where $P, S \subseteq \Sigma^*$). The entry at (p, s) is the weight $f(ps)$ returned by the membership oracle. Such tables amount mathematically to subblocks of the so-called *Hankel matrix*.

Once the table is *closed* and *consistent* in a suitable sense, the table can induce a WFA (Step 5 in Figure 6.1). The resulting WFA is examined in an equivalence query. If the oracle's answer is not `Eq.`, the oracle returns a counterexample word w as a witness of non-equivalence, which is in turn used for further expanding the table.

6.1.4 Closedness and Consistency in L*

The technical core of L*-style algorithms is expanding a table so that it is closed and consistent (Step 2-4 in Figure 6.1). The bottom line here is that a row of a table corresponds to a state in a WFA. *Closedness* is about having *enough rows*: it requires that, from each row, its "successor states" should be representable using the existing states (i.e. rows). Here *representable* means *identical* in the original L* for DFAs, and *linearly dependent* in the extensions for WFAs. *Consistency* of a table requires that such representations are consistent with the transition behaviors. Another way of putting it is that consistency means that the "successor state" is unique for each row.

Different L*-style algorithms use different mechanisms to ensure consistency. The original L* [6] conducts explicit check of both closedness and consistency. It adds rows if the table is not closed (i.e. not enough rows). When the table is not consistent, It means that successor states are ambiguous in the currently exposed transition relations. To resolve this ambiguity, the algorithm adds new columns.

In the semiring-weighted setting, consistency is hard to check directly, due to the use of linear combinations for state representation. Therefore the WFA learning algorithms in [12, 129] employ alternative indirect measures inspired by or similar to [87]. These measures consist of clever handling of counterexamples (Step 6 in Figure 6.1).

6.1.5 Consistency Issue in the Max-Plus WFA Learning

For the max-plus semiring, we first tried an implementation of the semiring-generic algorithm in [129], and unexpectedly found that some hypothesis automata A' are *unfaithful*, in the sense that they return a weight $f_{A'}(w)$ that is different from the weight $m(w)$ that the membership oracle answered previously.[*1] Here *hypothesis automata* refer to those in Step 5 in Figure 6.1, that is, those which are induced by tables that may not yet have passed the equivalence check.

In other words, in the algorithm from [129], some hypothesis automata may be unfaithful to the tables that induce them. We found its cause to be in the *failure of consistency*, which in turn is because the notion of linear (in)dependence is fragile for the max-plus semiring. For example, even if a row vector r_* is linearly *independent* of row vectors $r_1, \ldots, r_m$, the vector r_m can be linearly *dependent* on $r_*, r_1, \ldots, r_{m-1}$.

[*1] We note that the focus of the paper [129] is on the termination of learning (i.e. whether tables can be closed in finite steps), which is orthogonal to ours. It does not claim the faithfulness of hypothesis automata; thus this chapter does not show any of their results to be wrong.

This potential unfaithfulness of hypothesis automata A' does not mean that the algorithm in [129] can give a wrong final answer: in fact, words w such that $f_{A'}(w) \neq m(w)$ will be eventually identified by the equivalence oracle e as counterexamples, and A' will be fixed accordingly. However, complete equivalence oracles are rare in reality, which makes this "ultimate correctness" argument unrealistic. Given the possibility of using a hypothesis automaton A' itself as a surrogate model (without A' passing the equivalence check), A' being unfaithful even to previous membership queries should better be avoided. Moreover, the algorithm in [129] relies on the equivalence oracle for resolution of unfaithfulness, but (precise or approximate) equivalence queries are usually expensive.

Our countermeasure is to identify the notion of *column-closedness* as a substitute for consistency, and to enforce column-closedness in the course of L*-style WFA learning (which excludes potential unfaithfulness). We call the usual closedness notion *row-closedness* for distinction. Column-closedness is a clean algebraic condition that is dual to row-closedness. This cleanness makes the notion a pleasantly robust one, allowing a simple equational proof of the faithfulness of hypothesis automata for an arbitrary semiring. Column-closedness can be checked as efficiently as row-closedness.

6.1.6 Contributions

We summarize our contributions.

- In the setting of WFAs over the max-plus semiring, we find that the WFA learning algorithm in [129] can return hypothesis automata that are unfaithful to previous membership queries (Section 6.3.3).
- We introduce the notion of *column-closedness* that replaces consistency. We also present a WFA learning algorithm based on it, which works for an arbitrary semiring. We prove that the learned WFA is always faithful (Section 6.3).
- Focusing on the max-plus semiring, we further study properties and extensions of the algorithm (Section 6.4). We show 1) that our algorithm may not terminate or yield a minimal WFA and identify the reason, 2) that "best-effort" minimization is still possible, and 3) that we can make it noise-tolerant by relaxing equality constraints.

Some proofs are deferred to the appendix.

6.1.7 Notations

We use NumPy-like notations: for a matrix A, $A(p,:)$ is its p-th row and $A(:,s)$ is its s-th column. For a set of words $P \subseteq \Sigma^*$ and a character $\sigma \in \Sigma$, we define $\sigma P := \{\sigma w \mid w \in P\} \subseteq \Sigma^*$, and similarly $P\sigma := \{w\sigma \mid w \in P\}$. For a word w, w_i stands for the i-th character of w.

6.1.8 Related Work

Most of related work have been discussed so far. The WFA learning algorithms in [12, 129] are discussed in detail and compared with our algorithm in Section 6.3.3. Here we cover other pieces of work. In addition to the class of automata mentioned in §6.1, L*-style automata learning have been extended to various type of automata, such as timed automata [59], nominal automata [95], and symbolic automata [44]. L*-style automata learning algorithms are also applied to approximate *recurrent neural networks* for explainability and acceleration [7, 103, 134].

6.1.9 Organization of the Chapter

In Section 6.2, we give basic definitions. In Section 6.3, we discuss properties of weighted finite automata on semirings and present our algorithm. In Section 6.4, we discuss mathematical properties of the L*-style automata learning on the max-plus semiring. In Section 6.5, we conclude the chapter. In Section 6.6, we discuss the future work.

6.2 Preliminaries

6.2.1 The Max-Plus Semiring

In this chapter, we use the max-plus semiring whose base set consists of real numbers. This choice is inessential and our results apply to other base sets such as $\mathbb{Z} \cup \{-\infty\}$ and $\mathbb{Q} \cup \{-\infty\}$. The notation $\mathbb{R}_{\max}$ is standard in the community; see e.g. [1]. Refer to [45] for the general theory of semirings and semimodules.

Definition 6.1 (the max-plus semiring $\mathbb{R}_{\max}$) *The max-plus semiring* $\mathbb{R}_{\max}$ is a semiring defined as follows.

- The base set is $\mathbb{R} \cup \{-\infty\}$, where $-\infty$ is a symbol denoting the negative infinity.
- (Addition $\oplus_{\mathbb{R}_{\max}}$) $a \oplus_{\mathbb{R}_{\max}} b$ is $\max_{\mathbb{R}}(a,b)$ if $a, b \in \mathbb{R}$; it is a if $b = -\infty$; and it is b if

$a = -\infty$.

- (Multiplication $\otimes_{\mathbb{R}_{\max}}$) $a \otimes_{\mathbb{R}_{\max}} b = a +_{\mathbb{R}} b$ if $a, b \in \mathbb{R}$; $a \otimes_{\mathbb{R}_{\max}} b = -\infty$ otherwise.
- (Units) $0_{\mathbb{R}_{\max}} = -\infty$, and $1_{\mathbb{R}_{\max}} = 0_{\mathbb{R}}$.

The subscripts $\bullet_{\mathbb{R}_{\max}}$ may be omitted when they are clear from the context. Obviously, $\mathbb{R}_{\max}$ is commutative.

Over the max-plus semiring, a particular form of linear equations can be solved efficiently [22]. The procedure consists of 1) a solution candidate—it is called a *principal solution*, following the convention in the community [22], but we emphasize that a principal solution *may not be a solution*—and 2) checking whether the candidate is indeed a solution or not. This procedure is known to be complete: if the principal solution turns out to be not a solution, then the linear equation has no solution.

Definition 6.2 (principal solution) Consider a system of linear equations $xA = b$, where $A \in \mathbb{R}_{\max}^{n \times m}$ is a coefficient matrix, $b \in \mathbb{R}_{\max}^{m}$ is a row vector and $x \in \mathbb{R}_{\max}^{n}$ is an indeterminate row vector. Its *principal solution* is defined by

$$x(i) = \begin{cases} \min_{j \in [1,m]}(b(j) - A(i,j)) & \text{if } A(i,:) \neq (-\infty, \ldots, -\infty) \\ -\infty & \text{if } A(i,:) = (-\infty, \ldots, -\infty) \end{cases} \quad \text{for } i = 1, \ldots, n.$$

□

Although the usual definition (e.g. in [22]) assumes that no row of A is $(-\infty, \ldots, -\infty)$, we do not do so; instead we add the second clause as a workaround. Notice that this second clause resolves the apparent problem in min in the first clause: if $A(i,j) = -\infty$ then $b(j) - A(i,j)$ should be $+\infty$ that does not belong to $\mathbb{R}_{\max}$; but the condition $A(i,:) \neq (-\infty, \ldots, -\infty)$ ensures that there is some j for which $b(j) - A(i,j)$ is well-defined.

We note that multiple solutions may exist for $xA = b$. It is known that, if the principal solution is a solution and there is no row filled with $-\infty$ in A, then it is the maximum among the solutions in the pointwise order. It is also obvious that a system of the form $XA = B$ can be solved, where X, A, B are matrices, by solving $X(k,:) \cdot A = B(k,:)$ for each k [36].

There are several definitions known for linear independence in the max-plus semiring (they are not mutually equivalent). We find the following one suited for our purpose.

Definition 6.3 (weak linear (in)dependence [36]) Let M be a semimodule over a commutative semiring $\mathbb{S}$, and $X \subseteq M$ be a subset. X is *weakly linearly dependent* if there exists an element $x \in X$ such that x can be expressed as a linear combination $\bigoplus_\lambda c_\lambda \otimes x_\lambda$ of elements $x_\lambda \in X \backslash \{ x \}$. If X is not weakly linearly dependent, it is said to be *weakly linearly independent.*
□

6.2.2 Weighted Automata

Definition 6.4 (WFA) A *weighted finite automaton (WFA)* over a semiring $\mathbb{S}$ is a tuple $(\Sigma, \alpha, \beta, (A_\sigma)_{\sigma\in\Sigma})$, where Σ is a finite set (*alphabet*), α is a row vector in $\mathbb{S}^d$ (*initial vector*), β is a column vector in $\mathbb{S}^d$ (*final vector*), and A_σ is a matrix of size $d \times d$ called *transition matrix* of $\sigma \in \Sigma$. The size d of the vectors and matrices is called *the number of states* and denoted by $|A|$. □

Definition 6.5 (weighted language induced from WFA) A WFA $A = (\Sigma, \alpha, \beta, (A_\sigma)_{\sigma\in\Sigma})$ induces a function $f_A \colon \Sigma^* \to \mathbb{S}$ defined by $f_A(w_1 \dots w_n) = \alpha A_{w_1} \dots A_{w_n} \beta$. This function f_A is called the *weighted language* induced by A.

A weighted language that is induced by some WFA is said to be *rational*. For a WFA A, if there is no WFA that has the same weighted language and has a smaller number of states than A, then A is said to be *minimal*. □

We use matrices whose rows and columns are indexed by words. For label sets $P, S \subseteq \Sigma^*$ and a matrix $A \in \mathbb{S}^{P\times S}$, $A(p, s) \in \mathbb{S}$ is the entry at the p-th row and the s-th column.

The *Hankel matrix* of a language L is a mathematical construct that is behind L*-style automata learning. Its p-th row, for $p \in \Sigma^*$, is the left-derivative of L by p.

Definition 6.6 (Hankel matrix) Let Σ be an alphabet and $\mathbb{S}$ be a semiring. *A Hankel matrix* H is a matrix $\mathbb{S}^{\Sigma^*\times\Sigma^*}$ such that $H(r\sigma, s) = H(r, \sigma s)$ for any $r, s \in \Sigma^*$ and $\sigma \in \Sigma$. A weighted language $f \colon \Sigma^* \to \mathbb{S}$ induces a Hankel matrix, denoted by $H_f \in \mathbb{S}^{\Sigma^*\times\Sigma^*}$, by $H_f(p, s) = f(p \cdot s)$ for words $p, s \in \Sigma^*$. □

Hankel matrices are of infinite size, so algorithms use the following finite restrictions. In particular, Hankel subblocks correspond to the tables in L*-style algorithms (Figure 6.1).

Definition 6.7 (Hankel mask, Hankel subblock) For an alphabet Σ, a *Hankel mask* (P, S) is a pair of subsets of Σ^*. The *Hankel subblock* of a Hankel matrix H masked by (P, S), denoted by $H_{(P,S)}$, is the matrix $\mathbb{S}^{P\times S}$ such that $H_{(P,S)}(p, s) = H(p, s)$ for any $p \in P, s \in S$. □

6.3 WFA Learning for General Semirings

Here we introduce our algorithm, in which our new notion of column-closedness replaces consistency (see Section 6.1). We present the algorithm for a general semiring $\mathbb{S}$, and prove its "correctness" (we say *faithfulness*), i.e. that the learned WFA respects the weights given by the

membership oracle. Later in Section 6.3.3 we show the problem with the existing algorithm from [129] how it is resolved.

6.3.1 Row-Closedness and Column-Closedness

Definition 6.8 (row-closedness, column-closedness) Let $H \in \mathbb{S}^{\Sigma^* \times \Sigma^*}$ be a Hankel matrix, $P \subseteq \Sigma^*$ be prefix-closed, and $S \subseteq \Sigma^*$ be suffix-closed. The Hankel subblock $H_{(P,S)}$ is *row-closed* if the system of linear equations $X_\sigma H_{(P,S)} = H_{(P\sigma,S)}$ has a solution for any $\sigma \in \Sigma$. Here $X_\sigma \in \mathbb{S}^{P \times P}$ is the indeterminate matrix. Dually, $H_{(P,S)}$ is *column-closed* if the system of linear equations $H_{(P,S)} Y_\sigma = H_{(P,\sigma S)}$ has a solution for any $\sigma \in \Sigma$. Here $Y_\sigma \in \mathbb{S}^{S \times S}$ is the indeterminate matrix. □

These X_σ and Y_σ are used in the whole section.

Row-closedness means that each row of $H_{(P\sigma,S)}$ is a linear combination of the rows of $H_{(P,S)}$; this is the usual condition of *closedness* used in WFA learning [12, 129]. Our notion of column-closedness is its dual, meaning that each column of $H_{(P,\sigma S)}$ is a linear combination of the columns of $H_{(P,S)}$. Note that, in column-closedness, the column-index set S is extended *backward* to σS.

Lemma 6.9. *Let $H_{(P,S)}$ be row-closed and column-closed, and X_σ and Y_σ be solutions of the equations in Definition 6.8. Then we have $X_\sigma H_{(P,S)} = H_{(P,S)} Y_\sigma$.*

Proof. $X_\sigma H_{(P,S)} = H_{(P\sigma,S)} = H_{(P,\sigma S)} = H_{(P,S)} Y_\sigma$. ■

Lemma 6.10 (shifting)**.** *Let $P \subseteq \Sigma^*$ be prefix-closed, $S \subseteq \Sigma^*$ be suffix-closed, and $s, w \in \Sigma^*$ be such that $sw \in S$. If the Hankel subblock $H_{(P,S)}$ is row-closed, then we have $(X_{s_1} \dots X_{s_n} H_{(P,S)})(:, w) = H_{(P,S)}(:, sw)$. Dually, let $w, p \in \Sigma^*$ be such that $wp \in P$. If $H_{(P,S)}$ is column-closed, then we have $(H_{(P,S)} Y_{p_1} \dots Y_{p_n})(w, :) = H_{(P,S)}(wp, :)$.* □

Proof. We prove the first half of Lemma 6.10 by induction on the length of s. If s is empty, it is obvious. If s is nonempty, because S is suffix-closed, $s_2 \dots s_n w \in S$ holds, and thus, we have $(X_{s_2} \dots X_{s_n} H_{(P,S)})(:, w) = H_{(P,S)}(:, s_2 \dots s_n w)$ by the induction hypothesis. By definition, $X_{s_1} H_{(P,S)}(:, s_2 \dots s_n w) = H_{(Ps_1,S)}(:, s_2 \dots s_n w)$ holds, and we have $X_{s_1} H_{(P,S)}(:, s_2 \dots s_n w) = H_{(Ps_1,S)}(:, s_2 \dots s_n w) = H_{(P,s_1 S)}(:, s_2 \dots s_n w) = H_{(P,S)}(:, sw)$. The proof for the second half of Lemma 6.10 is similar. ■

Theorem 6.11 (WFA construction)**.** *Let H be a Hankel matrix, $P \subseteq \Sigma^*$ be prefix-closed, and $S \subseteq \Sigma^*$ be suffix-closed. Assume further that $H_{(P,S)}$ is both row-closed and column-closed. Let $A = (\Sigma, \alpha, \beta, A_\sigma)$ be the following WFA: $\alpha = (0, -\infty, \dots, -\infty)$ (0 is at the position ε),*

$\beta = H_{(P,S)}(:, \epsilon)$, *for* $\sigma \in \Sigma$, A_σ *is a solution of* $X_\sigma H_{(P,S)} = H_{(P\sigma,S)}$. *Then for* $p \in P, s \in S$, *we have* $f_A(ps) = H_{(P,S)}(p, s)$.

Note that the above construction of the WFA A only requires row-closedness to go through—it uses X_σ but no Y_σ. However, for the correctness property $f_A(ps) = H_{(P,S)}(p, s)$ (that the constructed WFA returns the value recorded in the Hankel subblock), we need column-closedness. This is seen in the proof below, and the necessity of the column-closedness is shown in Section 6.3.3 with an example.

Proof. Let $|p| = n$ and $|s| = m$. We write e_w for $(-\infty, \dots, 0, \dots, -\infty)$ (0 is at the position w), for $w \in \Sigma^*$.

$$\begin{aligned} f_A(ps) &= e_\epsilon X_{p_1} \dots X_{p_n} X_{s_1} \dots X_{s_m} H_{(P,S)}(:, \epsilon) \\ &\overset{\text{Lemma6.10}}{=} e_\epsilon X_{p_1} \dots X_{p_n} H_{(P,S)}(:, s) \\ &= e_\epsilon X_{p_1} \dots X_{p_n} H_{(P,S)} e_s^\top \\ &\overset{\text{Lemma 6.9}}{=} e_\epsilon H_{(P,S)} Y_{p_1} \dots Y_{p_n} e_s^\top \\ &= H_{(P,S)}(\epsilon, :)\, Y_{p_1} \dots Y_{p_n} e_s^\top \\ &\overset{\text{Lemma 6.10}}{=} H_{(P,S)}(p, :)\, e_s^\top = H_{(P,S)}(p, s). \end{aligned}$$

■

We note the simplicity of the above proof. It uses algebraic laws from Lemma 6.9–6.10, and does not rely on any matrix constructions such as rank factorization. This is in contrast with existing proofs such as [12, Theorem 3].

6.3.2 Generic WFA Learning Algorithm

The theoretical development in Section 6.3.1 yields an L*-style WFA learning algorithm in a straightforward manner: we add the check of column-closedness to the algorithm in [12, 129]. Fortunately, column-closedness is a linear-algebraic dual to row-closedness (Definition 6.8), so checking it is no harder than row-closedness.

Our algorithm is in Algorithm 6. It follows the outline in Figure 6.1, where Step 4 (check of closedness and consistency) is replaced by check of row- and column-closedness. It expands the table (i.e. the Hankel mask (P, S)), filling the entries by issuing membership queries, until the Hankel subblock $H_{(P,S)}$ is row- and column-closed (Line 15–18).

The Enclose-Row procedure for checking and enforcing row-closedness is much like in the existing algorithms [12], where the $p\sigma$-th row is checked if it is a linear combination of existing rows, and if the answer is no, it is added to the row set P.

The Enclose-Column procedure is totally dual to Enclose-Row. If the closedness check fails, a new word σs is added to the column set S.

The feasibility of the Enclose-Row and Enclose-Column procedures depends on

whether the equation in Line 25 and the corresponding equation in Enclose-Column can be solved, more precisely, whether we can detect the existence of a solution. We can do so efficiently with the max-plus semiring, as we discussed in Section 6.2.1. Once $H_{(P,S)}$ is seen to be row- and column-closed, a WFA is constructed by the recipe in Theorem 6.11 (Line 19 in Algorithm 6). When the constructed WFA is not the answer, the equivalence oracle returns a counterexample word w. The way in which w is handled (Step 6 in Figure 6.1) is the same as in [12, 87, 129], adding all relevant suffixes of w to the column set S (Line 9–11).

As in the previous work [12, 129], our algorithm terminates on some semrings such as the real semiring and PIDs. However, it is not guaranteed to terminate on an arbitrary semiring—for example, our algorithm may diverge in learning a WFA over the max-plus semiring. We believe that this non-termination is due to the nature of the underlying semirings, instead of the choice of an algorithm. Even if the linear equations in Enclose-Row and Enclose-Column are feasible (Line 25 and the corresponding line in Enclose-Column), there is no guarantee that finite repetitions of them lead to a row- and column-closed table. This is indeed the case with the max-plus semiring, as we discuss in Section 6.4.1. Another issue is that the resulting WFA is not necessarily minimal (see Section 6.4.1).

We conclude with the following correctness theorem.

Definition 6.12 (faithfulness) Let $m\colon \Sigma^* \to \mathbb{S}$ be a membership oracle, and $w^{(1)}, \dots, w^{(K)}$ be words (intuitively, these are the membership queries that have been issued). We say that a WFA A is *faithful* to m on $w^{(1)}, \dots, w^{(K)}$ if $f_A(w^{(k)}) = m(w^{(k)})$ for each $k \in [1, K]$.

Theorem 6.13. *In an execution of Algorithm 6, each hypothesis WFA A_{ext} produced in Line 3 is faithful to the oracle m, on the words $w^{(1)}, \dots, w^{(K)}$ that were queried to m previously during the algorithm's execution.* □

Proof. Note first that each $w^{(k)}$ can be written as $p \cdot s$ using elements $p \in P$ and $s \in S$ (Line 37). We have $f_A(ps) = \textsc{Subblock}(P, S)(p, s)$ by Theorem 6.11 (this uses column-closedness; see also Line 19). The latter is equal to $m(p \cdot s)$ by Line 37. ■

6.3.3 Comparison with Other WFA Learning Algorithms

Van Heerdt et al.'s algorithm [129] and our algorithm have the same goal, but they differ in two techniques: (1) When a counterexample w at Line 8 is reflected in the Hankel submatrix, our algorithm uses the *Balle-inspired refinement* [12]: take a prefix p of the counterexample w, and add all the suffixes of s into the column set S to keep S compact (Line 9-11). On the

Table 6.2: A Hankel submatrix made from the WFA A (left) and the next rows of the Hankel submatrix (right)

$H_{(P,S)}$	ϵ	a
ϵ	13	26
a	26	34
ab	35	40
b	28	30

$H_{(P\{a,b\},S)}$	ϵ	a	comb.
aa	34	42	$8 \otimes a$
aba	40	48	$14 \otimes a$
abb	44	49	$9 \otimes ab$
ba	30	39	$13 \otimes \epsilon \oplus 2 \otimes b$
bb	37	42	$2 \otimes ab$

other hand, van Heerdt et al.'s algorithm uses the *Maler-Pnueli-inspired refinement* [87]: add all the suffixes of the counterexample w into the column set instead of Line 9-11 in Algorithm 6. (2) Our algorithm extends the Hankel submatrix so that it becomes column-closed (Line 17), but van Heerdt et al.'s algorithm does not need this step.

As van Heerdt et al.'s algorithm does not ensure the column-closedness at the moment of the WFA construction (Line 12), the constructed WFA A_{ext} might be unfaithful to the previous answers of membership queries, i.e., there can be a word w such that $m(w) \neq f_{A_{\mathrm{ext}}}(w)$ and the membership query of w has already been issued. Even if the Hankel submatrix is not column-closed, the following property holds for the constructed WFA[*2]:

Theorem 6.14 (constructed WFA in [129])**.** *Let H, P, S, A be the same as in Theorem 6.11. If $H_{(P,S)}$ is row-closed, then we have $f_A(s) = H_{(P,S)}(\epsilon, s)$ for $s \in S$.*

The proof is almost the same as the proof of Theorem 6.11. By this property and the way the column set S is updated, the WFA A_{ext} is updated to satisfy $f_{A_{\mathrm{ext}}}(w) = f_A(w)$ for the counterexample w, and the learning successfully progresses.

One might expect that using Balle-inspired refinement and omitting to ensure the column-closedness will keep S compact and accelerate the learning. However, this "hybrid" method does not actually work. There exists a target WFA A that is learnable both by our algorithm and by van Heerdt et al.'s algorithm, but makes this hybrid method diverge. The WFA $A = (\{a, b\}, \alpha, \beta, (A_\sigma)_{\sigma\in\Sigma})$ with $\alpha_A = (6, 11, 1)$, $\beta_A = (7, 0, 6)^\top$, $A_a = ((2, 3, 1), (2, 0, 9), (3, 0, 8))^\top$, $A_b = ((9, 6, 2), (10, 3, 2), (8, 5, 4))^\top$ is such an example. Assume that the Hankel submatrix is as shown in the Table 6.2 and A_{ext} is the WFA constructed from the Hankel submatrix at Line 4 in Algorithm 6. This Hankel matrix is

[*2] This property is pointed out by an anonymous reviewer of AAAI20, which we were not aware of at first. We thank the anonymous reviewer for the useful comment.

row-closed, but not column-closed (the "comb." column shows that it is row-closed). In this situation, $f_A(ab) = H_{(P,S)}(ab, \epsilon) = 35$ but $f_{A_{\mathrm{ext}}}(ab) = 36$. If the answer of the equivalence query `res` at Line 5 is ab, then s is set to ϵ at Line 10 and S is not updated at Line 11. In total, the Hankel matrix is not updated and A_{ext} is also not updated. Then, the equivalence query e return ab at Line 5, and this leads to the infinite loop. The problem is that the counterexample ab is not reflected to the Hankel matrix, so we have to choose either our algorithm or van Heerdt et al.'s algorithm to make the learning progress.

When we turn to the real semiring, the algorithm in [12] does not explicitly check column-closedness nor consistency. The resulting WFA is nevertheless faithful (Definition 6.12). This is because column-closed is automatically maintained in the following way. Since Enclose-Row adds a row $H(P\sigma, S)$ only when it is not a linear combination of the rows of $H_{(P,S)}$, the Hankel subblock $H_{(P,S)}$ is maintained to be full-rank and landscape (meaning $|P| \leq |S|$). Hence the subspace generated by the columns of $H_{(P,S)}$ coincides with the whole space $\mathbb{R}^P$, and thus in particular, any column of $H_{(P,\sigma S)}$ is expressed as a linear combination of the columns of $H_{(P,S)}$. The latter means that it is column-closed.

In the max-plus setting, however, the Enclose-Row procedure does not maintain column-closedness. For illustration, observe that the ab-th row of $H_{(P,S)}$ in (6.1) cannot be expressed as a linear combination of the ϵ-th and a-th rows, but the a-th row is $8 \otimes (\epsilon\text{-th row}) \oplus (-9) \otimes (ab\text{-th row})$. This example shows that even if a new row cannot be expressed as a linear combination of the existing rows, adding the row can destroy the weak linear independency of the matrix (Definition 6.3).

6.4 Further on the Max-Plus WFA Learning

6.4.1 Non-Termination and Non-Minimality

In general, our Algorithm 6 does not terminate: even if the key conditions in Enclose-Row and Enclose-Column are effectively checkable (Line 25 and the corresponding line in Enclose-Column), we may never get the Hankel subblock row- or column-closed.

$P \setminus S$	ϵ	b	c	$\cdots$
ϵ	0	0	0	$\cdots$
a	0	1	2	$\cdots$
aa	0	2	4	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$
a^n	0	n	$2 \times n$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

To see it, consider the following example: for the WFA $B = (\{a, b, c\}, \alpha_B, \beta_B, (B_\sigma)_{\sigma\in\Sigma})$ with $\alpha_B = (0, 0, 0)$, $\beta_B = e_1$,

$$B_a = \begin{pmatrix} e_1 \\ 1 \otimes e_2 \\ 2 \otimes e_3 \end{pmatrix}, B_b = \begin{pmatrix} -\infty \\ e_1 \\ -\infty \end{pmatrix}, B_c = \begin{pmatrix} -\infty \\ -\infty \\ e_1 \end{pmatrix}, \qquad (6.1)$$

where $-\infty$ means the row filled with $-\infty$. The Hankel matrix of f_B is as shown on the right. It is not hard to see that no choice of finite rows generates the semimodule spanned by all the rows. Therefore Hankel subblocks are never row-closed from a certain moment on (specifically after $b, c \in S$), making Algorithm 6 diverge.

However, this does not mean that the semimodule spanned by all the rows is not finitely generated. In fact, since the Hankel matrix comes from the 3-state WFA B, there is a canonical choice of generators r_1, r_2, r_3 that span all the rows: $r_i = \lambda w.e_i B_{w_1} \dots B_{w_{|w|}} \beta_B$ for $i = 1, 2, 3$.

Once we find generators spanning the rows of a Hankel subblock, we can construct a WFA that is compatible with the Hankel subblock. This is by the following procedure. (The problem is how to find such row vectors. L*-style algorithms search only in rows of the Hankel matrix, and for the max-plus semiring this is not enough, as shown by the above example B.)

For a Hankel matrix H over the max-plus semiring, if there are vectors $r_1, \dots, r_n \in \mathbb{R}_{\max}^{\Sigma^*}$ such that (1) the row $H(w, :)$ is expressed as $c_{w,1} \otimes r_1 \oplus \dots \oplus c_{w,n} \otimes r_n$ for any $w \in \Sigma^*$, and (2) there exists a homomorphism $\varphi_\sigma \colon \mathbb{R}_{\max}^n \to \mathbb{R}_{\max}^n$ for any $\sigma \in \Sigma$ such that $\varphi(c_{w,1}, \dots, c_{w,n}) = (c_{w\sigma,1}, \dots, c_{w\sigma,n})$ for any $w \in \Sigma^*$, then a WFA $A = (\Sigma, \alpha_A, \beta_A, (A_\sigma)_{\sigma \in \Sigma})$ which is consistent with H can be constructed as follows:

- $\alpha_A = (c_{\epsilon,1}, \dots, c_{\epsilon,n})$,
- $\beta_A = (r_1(\epsilon), \dots, r_n(\epsilon))^\top$,
- For $\sigma \in \Sigma$, A_σ is the matrix induced from φ_σ.

The validity of the WFA is easy:

$$\begin{aligned}
f_A(w) &= \alpha_A A_{w_1} \dots A_{w_m} \beta_A \\
&= (c_{\epsilon,1}, \dots, c_{\epsilon,n}) A_{w_1} \dots A_{w_m} \beta_A \\
&= (c_{w_1,1}, \dots, c_{w_1,n}) A_{w_2} \dots A_{w_m} \beta_A \\
&= \dots \\
&= (c_{w,1}, \dots, c_{w,n}) \beta_A \\
&= (c_{w,1}, \dots, c_{w,n}) \begin{pmatrix} r_1 \\ \vdots \\ r_n \end{pmatrix} e_\epsilon^\top \\
&= H(w, :) e_\epsilon^\top \\
&= H(w, \epsilon).
\end{aligned}$$

The problem is that L*-style algorithms (including Algorithm 6) restricts its search for generators of the *rows of the Hankel matrix*. Lifting this restriction is at the cost of greater computational cost, and is a topic of future work.

6.4.2 Best-Effort Minimization

The example B in Section 6.4.1 is also an extreme witness that Algorithm 6 may not yield a minimal WFA: it yields infinitely many states while the minimal WFA has three states. A similar problem of finding a minimal representation of max-plus linear systems has been studied in control theory [41, 42]. In general, minimization of such systems is considered to be a computationally expensive problem.

Although it is not always possible to obtain the minimal WFA from Algorithm 6, we can still try to eliminate linearly dependent rows during the execution, so that we obtain a smaller WFA. When the rows of the Hankel subblock $H_{(P,S)}$ are weakly linearly dependent, we can reduce some rows and shrink the number of states of the constructed WFA in Theorem 6.11. The reduction of the Hankel subblock is possible just by repeatedly sweeping the rows and find a row that can be expreessed by the other rows and removing it. It is known that the order of removing the row does not affect the number of the resulting weakly linear denpendent rows (see Theorem 3.3.9 in [22]).

We assume that we get a WFA $A = (\Sigma, \alpha, \beta, (A_\sigma)_{\sigma\in\Sigma})$ as a result, and the Hankel subblock $H_{(P,S)}$ is weakly linearly dependent. In this section, we index the matrices by numbers as the usual matrix theory, and $(-\infty, \ldots, \overset{i\text{-th}}{\overset{\vee}{0}}, \ldots, -\infty)$ is abbreviated as e_i. Without loss of generality, we can assume that the first n rows of $H_{(P,S)}$ are weakly linearly independent. By the definition of weak linear dependency, for $i = n+1, \ldots, |P|$, $H_{(P,S)}(i,:)$ can be expressed as $c_{i1}H_{(P,S)}(1,:) \oplus \cdots \oplus c_{in}H_{(P,S)}(n,:)$ by coefficients $c_{i1}, \ldots, c_{in} \in \mathbb{R}_{\max}$. Hence, letting $C \in \mathbb{R}_{\max}^{|P|\times n}$ be

$$C = \begin{array}{r} {\scriptstyle n\text{-th} >} \\ {\scriptstyle (n+1)\text{-th} >} \\ \\ {\scriptstyle |P|\text{-th} >} \end{array} \begin{bmatrix} 0 & -\infty & \cdots & -\infty \\ \vdots & \ddots & \ddots & \vdots \\ -\infty & \cdots & -\infty & 0 \\ c_{n+1,1} & \cdots & \cdots & c_{n+1,n} \\ \vdots & \ddots & \ddots & \vdots \\ c_{|P|,1} & \cdots & \cdots & c_{|P|,n} \end{bmatrix}, \tag{6.2}$$

and $D \in \mathbb{R}_{\max}^{n\times|P|}$ be

$$D = \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix}, \tag{6.3}$$

$H_{(P,S)} = CDH_{(P,S)}$ holds (remark that CD is not the identity). Then the WFA $B = (\Sigma, \alpha C, D\beta, (DA_\sigma C)_{\sigma\in\Sigma})$ yields the same language f_A, and its number of states is n, which

is smaller than the number of states of A. We give the sketch of the proof by showing $f_B(\sigma\tau) = f_A(\sigma\tau)$ for a word $\sigma\tau \in \Sigma^*$. The idea is to reduce $CDH_{(P,S)}$ to $H_{(P,S)}$ by the relation above, and make the form of $CDH_{(P,S)}$ by using Lemma 6.9.

$$\begin{aligned}
f_B(\sigma\tau) &= (\alpha C)(DA_\sigma C)(DA_\tau C)(D\beta) \\
&= \alpha CDA_\sigma CDA_\tau CDH_{(P,S)}e_\epsilon^\top \\
&= \alpha CDA_\sigma CDA_\tau H_{(P,S)}e_\epsilon^\top \\
&= \alpha CDA_\sigma CDH_{(P,S)}B_\tau e_\epsilon^\top \\
&= \alpha CDA_\sigma H_{(P,S)}B_\tau e_\epsilon^\top \\
&= \alpha CDH_{(P,S)}B_\sigma B_\tau e_\epsilon^\top \\
&= \alpha H_{(P,S)}B_\sigma B_\tau e_\epsilon^\top \\
&= \alpha A_\sigma A_\tau H_{(P,S)}e_\epsilon^\top \\
&= \alpha A_\sigma A_\tau \beta \\
&= f_A(\sigma\tau).
\end{aligned}$$

6.4.3 Tolerating Noise and Numeric Errors

In most real-world applications, Algorithm 6 is too sensitive to noise and numeric errors. For a Hankel subblock to be row- or column-closed, we need suitable equalities to hold (namely Line 25 and the corresponding line in Enclose-Column), which would hardly hold under potentially noisy data and numeric errors.

We can make Algorithm 6 noise-tolerant, by replacing the above equality check with the following procedure. Let $A \in \mathbb{R}_{\max}^{m\times n}$ be a matrix, and $b \in \mathbb{R}_{\max}^n$. Then it is known [36] that we can effectively find a vector $x \in \mathbb{R}_{\max}^m$ minimizing $\max_i |(xA)_i - b_i|$. We can regard this x as an approximate solution. Using this x and checking whether $\max_i |(xA)_i - b_i|$ is below a pre-fixed threshold $\varepsilon > 0$ can replace the equality checks in Line 25 and the corresponding line in Enclose-Column.

6.5 Conclusions

We proposed the column-closedness as the condition to construct a faithful WFA from a Hankel subblock, and proved the validity in a semiring-generic way without using the specific linear algebra over fields. Based on this theory, we developed a new WFA learning method over general semirings including the max-plus semiring. We proved that the column-closedness is necessary to ensure the faithfulness of the constructed WFAs. We discussed the non-termination and the hardness of minimization by a concrete example over the max-plus

semiring, and identified the reason of them. We proposed a noise-tolerant variant of our algorithm over the max-plus semiring towards the application of the WFA learning in a realistic setting.

6.6 Future Work

As future work, it is interesting to identify the subclass of the rational weighted languages over the max-plus semiring on which van Heerdt et al.'s and our learning algorithms terminate. Investigating the practical performance of various WFA learning algorithms is also important. For experiments we have implemented van Heerdt et al.'s and our algorithms, but the evaluation of the implementations is left open. The program is available on `https://github.com/ERATOMMSD/tropical_learning_public`.

Algorithm 6 Our WFA learning algorithm over an arbitrary semiring $\mathbb{S}$. "Step n" refers to those in Figure 6.1. Given: Σ (alphabet), m (membership oracle), e (equivalence oracle)

procedure LEARN ▷ Main procedure
  $P \leftarrow \{\epsilon\}, S \leftarrow \{\epsilon\}$ ▷ Step 1
  $A_{\text{ext}} \leftarrow \text{EXTRACT}(P, S)$ ▷ Steps 2–5
  loop
    `res` $\leftarrow e(A_{\text{ext}})$
    if `res` $=$ `Eq.` **then**
      break
    else if `res` $= w$ for a word $w \in \Sigma^*$ **then** ▷ Step 6
      $p \leftarrow$ (the longest prefix of w within P)
      $s \leftarrow$ (the tail of w of length $|w| - 1 - |p|$)
        ▷ If the length is negative, it is empty
      $S \leftarrow S \cup$ (all the suffixes of s)
      $A_{\text{ext}} \leftarrow \text{EXTRACT}(P, S)$ ▷ Steps 2–5
  return A_{ext}

---

procedure EXTRACT(P, S)
  do
    $(P,$ `updatedR`?$) \leftarrow$ ENCLOSE-ROW(P, S)
    $(S,$ `updatedC`?$) \leftarrow$ ENCLOSE-COLUMN(P, S)
  while `updatedR`? or `updatedC`? is `updated`
  return WFA $(\Sigma, \alpha, \beta, (A_\sigma)_{\sigma\in\Sigma})$ of $\mathrm{H}(P, S)$
    ▷ The construction of Theorem 6.11

---

procedure ENCLOSE-ROW(P, S)
  `updated`? $\leftarrow$ `not_updated`
  loop
    `adding` $\leftarrow$ `nil`
    for $(p, \sigma) \in P \times \Sigma$ **do**
      if $\nexists x$ s.t. $x \cdot \mathrm{H}(P, S) = \mathrm{H}(\{p\sigma\}, S)$ **then**
        `adding` $\leftarrow p\sigma$
        break
    if `adding` $=$ `nil` **then**
      break
    else
      $P \leftarrow P \cup \{$ `adding` $\}$
      `updated`? $\leftarrow$ `updated`
  return $(P,$ `updated`?$)$ ▷ Now $H_{(P,S)}$ is row-closed

---

procedure ENCLOSE-COLUMN(P, S)
    ▷ Omitted as it is almost the same as ENCLOSE-ROW
  return $(S,$ `updated`?$)$ ▷ Now $H_{(P,S)}$ is column-closed

---

procedure H(P, S) ▷ Returns $H_{(P,S)}$
  return matrix $\lambda(p, s) \in P \times S.\quad m(p \cdot s)$
    ▷ Memoize answers of m and implicitly process Step 3

Chapter 7

Conclusions and Future Work

Abstraction is a method to combine multiple states of a target system and construct a simpler system, and makes it easy to apply formal methods to verify systems. The goal of this thesis was to looking at the algebraic structure behind a target system for efficient abstraction and to extend the range of systems that can be efficiently verified. By looking at such an algebraic structure, we can use the theorems holding in the algebraic structure to efficiently abstract states or to verify the abstracted system. Algebraic structures used in abstraction are typically vector spaces $\mathbb{R}^n$, the polynomial ring $\mathbb{R}[\vec{X}]$, the integer ring $\mathbb{Z}$ (discussed in Section 1.3.5). The work discussed in Chapters 3-6 are in this context. We are reviewing the conclusion and future work for each chapter, and discussing the possibility of combining them.

7.1 Reviewing Chapters 3-6

In Chapter 3, we developed a method to find polynomial interpolants by looking at the algebraic structure $\mathbb{R}[\vec{X}]$ based on Dai et al's work [39]. A challenge of their work is that, for a formulae A and B, if shapes described by A and B are "touching", their method cannot find any interpolant of A and B. Our work coped this challenge so that interpolants are constructed even if the shapes are "touching." An algebraic technique used in our work was that, in addition to ideals and cone structures known as substructures of $\mathbb{R}[\vec{X}]$, we proposed the *strict cone*. By constructing an SOS programming problem based on these structures, our motivating example was solved by using solvers, and our method successfully constructed polynomial interpolants for various "touching" problems. Another work [26] in this direction was also proposed after this work, and we believe that our work contributed to the validation of polynomial programs.

In Chapter 4, we studied bit-vector (BV) interpolant generation technique to verify a program in which an integer variable causes overflow and wraparound. This work is based on

Griggio's work [53], which generate bit-vector interpolants by converting a BV interpolant generation problem into a LIA interpolant generation and translating the LIA interpolant into a BV interpolant. The challenge of the previous work was that the translating back is incomplete and sometimes fails at generating BV interpolants. Our algebraic technique *boxing and gapping* wes developed by the hint that $(\mathbb{Z}/n\mathbb{Z})^d$, which models the memory space, forms a torus. The experimental result showed that our technique generates compact interpolants and accelerates the verification. As the overflow is a major cause of bugs and realistic, we believe this work contributes to the verification of realistic programs.

In Chapter 5, we approximated RNNs by WFAs, which are state machines with states in $\mathbb{R}^d$. This work was based on Weiss et al.'s work [134], which extracts a DFA from a target RNN by looking at the state space of the target RNN. We showed that a method to approximate RNNs with WFAs efficiently, and showed the applicability of the approximation to explainability. As the benefit of the algebraic property of WFAs, we can use the associative law of matrix multiplication. Gutiérrez et al. [56] showed that the associative law is useful to maximize the output of a WFA with an example of ours, and this shows the usefulness of the approximation.

In Chapter 6, we proposed the *faithful* WFA learning for the max-plus semiring, and proved that it is semiring generic. We also investigated WFA learning mathematically, and showed that the termination and minimality of L*-style WFA learnings over generic semirings cannot be guaranteed with the concrete example. This shows the limitation of L*-style WFA learnings over generic semirings, and that we have to modify the style of learning to guarantee the termination.

In this thesis, we made philosophy

> Look at an algebraic system whose underlying set corresponds to the state space of the system and whose operations describe the transition of the system. It leads us to find an efficient verification method.

and practiced it. Firstly, we proposed new verification techniques by using algebraic structures and their theorems (Chapters 3-4). Secondly, we proposed new techniques to approximate complex models into simple models so that the approximated models can be verified with formal methods (Chapters 5-6).

7.2 Future Work

As the future work of Chapter 3, we could use rigorous numerical methods to solve the numerical optimization problems in the process. We can transform the SDP problems in the process by using *facial reduction* [132] into simplified and equivalent SDP problems. As the

problems in the chapter were artificial and simple, making a good benchmark of programs with polynomials and developing our method based on the observations on the benchmark would be important. In a general point of view, this work was about going forward and backward between symbolic techniques, which provide exact correctness, and numeric techniques, which provide the quickness. Developing this kind of research could lead to scalable verification supported by numerical optimization. The opposite direction of this technique can also be interesting: an inaccurate and quick verification to support partial correctness of a target system.

As the future work of Chapter 4, we are curious about the performance of the Griggio's multi-layered interpolation combined with our method, as our method developed in the chapter was the improvement of one layer of the Griggio's multi-layered interpolation.

As the future work of Chapter 5, we can compare our method with other simplification methods of RNNs. It would be interesting to find an extension of our work using linear 2-RNNs as simplified models, which takes a sequence of $\mathbb{R}^{\Sigma}$ as input, instead of WFAs and automata learning for them [113]. This extension can be a good solution to the problem of scalability about the alphabet size. More practical experiments and evaluations are needed by using RNNs used in industry. Problems about the relationship between RNNs and DFAs still remain. For example, we can think of a problem like this hinted by interpolation problems: for an RNN R expressing a language $L \subset \Sigma^*$, find sequences $\{ A_n \}_n, \{ B_n \}_n$ of regular languages such that $A_n \subseteq L \subseteq B_n$ holds for any n and $\|A_n - B_n\| \to 0$ $(n \to \infty)$ for a norm.

As the future work of Chapter 6, it would be interesting to investigate what oracles in addition to membership query answerers and equivalence query answerers can guarantee the termination of the learning. The limitation also raises an interesting problem in formal language theory: what is the characterization of a rational language on max-plus semiring where L*-style learnings actually terminate?

Combining Chapter 5 and Chapter 6 and apply the automata learning on the max-plus semiring to RNNs will be also interesting, as there is a possiblity that the max operation can go along with ReLU functions in RNNs. In this research, we would need a fast library that enables us to manipulate the max-plus semiring.

From a high-level perspective, it will be needed to research programming languages that are compatible with program verification based on algebraic abstraction, not only to develop verification techniques.

There are two reasons why it is needed. Firstly, the complexity of program verification is sometimes due to simple operations are embedded into complex features, and we are not sure which part of programs definitely needs an advanced verification techniques. Encouraging programmers to write programs straightly by the design of languages should make program

verification easier and lead to identifying real needs of advanced techniques. For example, an integer variable is often used as a sequence of flags. If an integer variable is used in such a way in a program, the verifier needs to support bit-shifting and the modulo operation and the program gets unnecessarily complex for program verifiers. Secondly, it is known that the operations on the algebraic structures currently used in program verification are costly, and richening the expressivity does not seem to useful for industry programmers. Identifying important operations used in realistic programs and finding a subclass of algebraic structures induced from the operations could lead to applicable and mathematically-interesting techniques.

For the research of this direction, it would be a good starting to analyze the use cases of features of programming languages. Using the approach of software engineering and making questionnaires would be one way to do this. There is a machine learning technique to generate comments from a program [84], so applying such techniques can be another way.

Appendix A

Appendix of Chapter 5

A.1 Detailed WFAs Extracted from `wparen`

A.1.1 The WFA Extracted by $\mathbf{RGR}(5)$

Figure A.1 illustrates the WFA extracted from the RNN trained by `wparen` by $\mathbf{RGR}(5)$. The initial and final vectors, and the transition matrices are in Figure A.2 and A.5.

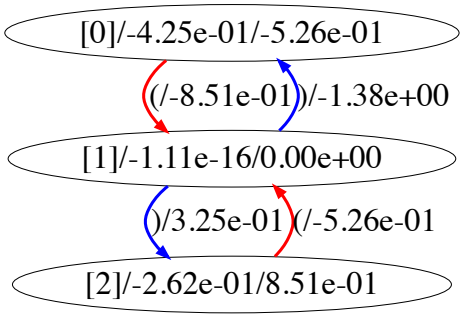


Figure A.1: The extracted WFAs by $\mathbf{RGR}(5)$

$$\alpha_A = \begin{pmatrix} -4.24538470e-01 \\ -1.10824765e-16 \\ -2.62446661e-01 \end{pmatrix}, \beta_A = \begin{pmatrix} -0.52582891 \\ 0 \\ 0.85059036 \end{pmatrix}$$

Figure A.2: The initial and accepting vector of the extracted WFAs by $\mathbf{RGR}(5)$

A.1.2 The WFA Extracted by **RGR**(15)

Figure A.3 illustrates the WFA extracted from the RNN trained by `wparen` by **RGR**(15). The initial and final vectors, and the transition matrices are in Figure A.4, A.6, and A.7.

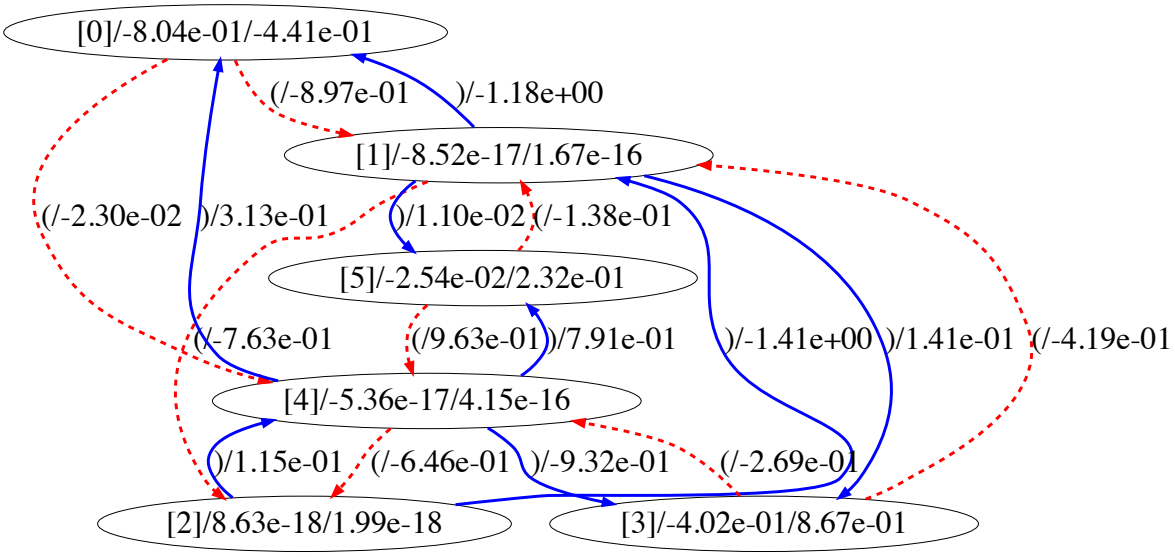


Figure A.3: The extracted WFAs by **RGR**(15)

$$\alpha_A = \begin{pmatrix} -8.03989494e-01 \\ -8.52463864e-17 \\ 8.62805331e-18 \\ -4.02164050e-01 \\ -5.35508802e-17 \\ -2.54365543e-02 \end{pmatrix}, \beta_A = \begin{pmatrix} -4.41053365e-01 \\ 1.66533454e-16 \\ 1.99459802e-18 \\ 8.67087989e-01 \\ 4.14765037e-16 \\ 2.31582271e-01 \end{pmatrix}$$

Figure A.4: The initial and accepting vector of the extracted WFAs by **RGR**(15)

$$A_(= \begin{pmatrix} 0.00000000e+00 & -8.50590360e-01 & 0.00000000e+00 \\ 0.00000000e+00 & -8.59785403e-17 & 0.00000000e+00 \\ 0.00000000e+00 & -5.25828907e-01 & 0.00000000e+00 \end{pmatrix}$$

$$A_) = \begin{pmatrix} 0.00000000e+00 & 0.00000000e+00 & 0.00000000e+00 \\ -1.37593037e+00 & 0.00000000e+00 & 3.25063688e-01 \end{pmatrix}$$

$$A_0 = \begin{pmatrix} 9.97100655e-01 & 0.00000000e+00 & -6.69023024e-04 \\ 2.91425732e-19 & 9.99289799e-01 & 1.63393525e-16 \\ 1.78230607e-03 & -3.85026199e-16 & 9.99284687e-01 \end{pmatrix}$$

$$A_1 = \begin{pmatrix} 9.97754767e-01 & 0.00000000e+00 & -1.10479143e-03 \\ 1.51485831e-19 & 1.00062565e+00 & 1.63529229e-16 \\ 9.26459427e-04 & -3.85540902e-16 & 1.00011463e+00 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 9.97342696e-01 & 0.00000000e+00 & -7.41831462e-04 \\ 2.47685043e-19 & 1.00090522e+00 & 1.63425319e-16 \\ 1.51479608e-03 & -3.85648619e-16 & 9.99479132e-01 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} 9.97152196e-01 & 0.00000000e+00 & -5.48790930e-04 \\ 2.62410313e-19 & 1.00054480e+00 & 1.63352215e-16 \\ 1.60485312e-03 & -3.85509751e-16 & 9.99032039e-01 \end{pmatrix}$$

$$A_4 = \begin{pmatrix} 9.97528034e-01 & 0.00000000e+00 & -9.46331754e-04 \\ 2.49455242e-19 & 1.00138911e+00 & 1.63510808e-16 \\ 1.52562229e-03 & -3.85835064e-16 & 1.00000197e+00 \end{pmatrix}$$

$$A_5 = \begin{pmatrix} 9.97514811e-01 & 0.00000000e+00 & -1.16262080e-03 \\ 1.67099898e-19 & 9.99657498e-01 & 1.63514942e-16 \\ 1.02195218e-03 & -3.85167873e-16 & 1.00002725e+00 \end{pmatrix}$$

$$A_6 = \begin{pmatrix} 9.98021793e-01 & 0.00000000e+00 & -1.24638991e-03 \\ 1.41062546e-19 & 9.99932885e-01 & 1.63603900e-16 \\ 8.62712533e-04 & -3.85273980e-16 & 1.00057130e+00 \end{pmatrix}$$

$$A_7 = \begin{pmatrix} 9.97351931e-01 & 0.00000000e+00 & -1.02957965e-03 \\ 2.34439773e-19 & 1.00063437e+00 & 1.63494750e-16 \\ 1.43379045e-03 & -3.85544261e-16 & 9.99903757e-01 \end{pmatrix}$$

$$A_8 = \begin{pmatrix} 9.97459514e-01 & 0.00000000e+00 & -7.96005125e-04 \\ 2.27400184e-19 & 1.00190830e+00 & 1.63446209e-16 \\ 1.39073761e-03 & -3.86035107e-16 & 9.99606890e-01 \end{pmatrix}$$

$$A_9 = \begin{pmatrix} 9.97267117e-01 & 0.00000000e+00 & -5.40868994e-04 \\ 3.80598316e-19 & 9.99770233e-01 & 1.63441973e-16 \\ 2.32766916e-03 & -3.85211310e-16 & 9.99580985e-01 \end{pmatrix}$$

Figure A.5: The transition matrices of the extracted WFAs by **RGR**(5)

$$A_(= \begin{pmatrix} -8.13455245e-17 & -8.97187248e-01 & -9.14353577e-17 & 6.05261178e-17 & -2.29558342e-02 & 2.63293240e-16 \\ 4.66228882e-33 & 4.35160982e-16 & -7.63485724e-01 & -8.48075160e-17 & 2.11401164e-16 & -2.35492323e-16 \\ -5.15551962e-33 & -4.68274918e-17 & 1.26662796e-16 & 1.62130062e-32 & 1.23274983e-17 & 4.28769662e-32 \\ -2.06614561e-17 & -4.19377765e-01 & 1.32499255e-16 & 4.76754407e-17 & -2.68850720e-01 & 3.12643472e-16 \\ -3.44414230e-32 & 1.23963211e-16 & -6.45824705e-01 & -7.17378037e-17 & 4.14160693e-16 & -1.99200529e-16 \\ -7.75638689e-17 & -1.38482243e-01 & 1.60067044e-15 & -6.32326215e-17 & 9.62908262e-01 & -6.69148936e-16 \end{pmatrix}$$

$$A_) = \begin{pmatrix} -5.54148872e-17 & -1.97207974e-20 & 2.03671709e-33 & 7.46642668e-18 & 1.61218726e-21 & -2.61486321e-19 \\ -1.18211303e+00 & 2.82084078e-16 & 2.75198255e-17 & 1.40589069e-01 & -3.46762135e-17 & 1.09968804e-02 \\ 1.29599806e-16 & -1.41195049e+00 & -5.84247592e-17 & 3.14911192e-18 & 1.15427817e-01 & 3.43913931e-16 \\ 2.33225409e-16 & -7.98892997e-17 & 4.54190736e-32 & 3.57233072e-17 & 6.53099913e-18 & -5.84636207e-17 \\ 3.13336842e-01 & -1.07422125e-15 & -7.71191144e-16 & -9.32439297e-01 & 4.01725110e-16 & 7.91163583e-01 \\ 1.01495616e-15 & -8.56767927e-16 & 6.25086145e-31 & 6.81033112e-16 & 7.00413022e-17 & -7.20640086e-16 \end{pmatrix}$$

$$A_0 = \begin{pmatrix} 9.99296632e-01 & -5.59800079e-17 & 4.40354106e-17 & 3.98075885e-04 & 6.13728232e-17 & -2.01408442e-04 \\ 1.15541074e-17 & 1.00082887e+00 & -1.21731926e-17 & -4.17682688e-18 & 7.78834199e-04 & -2.67091700e-16 \\ 2.98176861e-19 & -7.22564866e-16 & 1.00029933e+00 & 2.09488298e-16 & -2.31000082e-16 & 3.49208348e-16 \\ -1.56179092e-03 & 2.15918979e-16 & 6.43243081e-16 & 9.98434636e-01 & 2.04617242e-16 & -1.60951935e-03 \\ 1.17519561e-16 & 1.17507940e-03 & 6.82685692e-16 & -5.63812348e-17 & 1.00486186e+00 & -4.86957236e-16 \\ 1.32703493e-02 & -1.04785834e-15 & 3.03641615e-16 & 9.09412823e-05 & -1.50126714e-16 & 1.01293307e+00 \end{pmatrix}$$

$$A_1 = \begin{pmatrix} 9.99407998e-01 & -5.60117474e-17 & 4.39625414e-17 & 3.24682067e-04 & 6.14442452e-17 & -8.59144355e-05 \\ 1.15304960e-17 & 1.00102065e+00 & -1.22272583e-17 & -4.15177985e-18 & 6.91564023e-04 & -2.67083620e-16 \\ 1.30843906e-19 & -7.22766725e-16 & 1.00097799e+00 & 2.09749493e-16 & -2.31366661e-16 & 3.49447287e-16 \\ -1.64676070e-03 & 2.15504979e-16 & 6.43087712e-16 & 9.98109329e-01 & 2.04786438e-16 & -1.61269869e-03 \\ 1.17108223e-16 & 4.56507250e-03 & 6.82503884e-16 & -5.50183867e-17 & 1.00135870e+00 & -4.85110292e-16 \\ 9.32707190e-03 & -1.04832190e-15 & 3.06831309e-16 & 5.49548865e-03 & -1.45088795e-16 & 1.01367566e+00 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 9.99293546e-01 & -5.59120244e-17 & 4.40553625e-17 & 5.06854606e-04 & 6.14996619e-17 & -5.55034262e-05 \\ 1.15903828e-17 & 1.00119303e+00 & -1.22194261e-17 & -4.19667887e-18 & 4.98702032e-04 & -2.66997709e-16 \\ 2.26523367e-19 & -7.22509804e-16 & 1.00045605e+00 & 2.09575372e-16 & -2.31089602e-16 & 3.49280410e-16 \\ -1.60081145e-03 & 2.15324516e-16 & 6.42962343e-16 & 9.97884313e-01 & 2.04570862e-16 & -1.72690027e-03 \\ 1.17588225e-16 & 5.06284191e-03 & 6.82602341e-16 & -5.57205463e-17 & 1.00295379e+00 & -4.86382094e-16 \\ 1.15881399e-02 & -1.05019147e-15 & 3.05204965e-16 & 3.16454535e-03 & -1.46713771e-16 & 1.01380881e+00 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} 9.99281268e-01 & -5.59573383e-17 & 4.40180842e-17 & 3.78557978e-04 & 6.13506059e-17 & -1.84068466e-04 \\ 1.16388481e-17 & 1.00142057e+00 & -1.23070729e-17 & -4.17728883e-18 & 1.77934070e-04 & -2.66818663e-16 \\ 2.71166106e-19 & -7.23094524e-16 & 1.00097409e+00 & 2.09527252e-16 & -2.31028483e-16 & 3.49375582e-16 \\ -1.70720401e-03 & 2.15879244e-16 & 6.43231698e-16 & 9.98412235e-01 & 2.04758028e-16 & -1.50939736e-03 \\ 1.17289083e-16 & 7.79522548e-05 & 6.83082252e-16 & -5.65721181e-17 & 1.00553729e+00 & -4.87459412e-16 \\ 1.29556596e-02 & -1.04791865e-15 & 3.03922900e-16 & -7.47388994e-04 & -1.51257632e-16 & 1.01167398e+00 \end{pmatrix}$$

$$A_4 = \begin{pmatrix} 9.99272730e-01 & -5.58843634e-17 & 4.40804524e-17 & 5.68760210e-04 & 6.15529753e-17 & -2.10161070e-05 \\ 1.15820152e-17 & 1.00117000e+00 & -1.21945297e-17 & -4.20586579e-18 & 6.86122171e-04 & -2.67119334e-16 \\ 1.91336929e-19 & -7.22479004e-16 & 1.00061898e+00 & 2.09693820e-16 & -2.31351687e-16 & 3.49382751e-16 \\ -1.30996389e-03 & 2.15029536e-16 & 6.42794941e-16 & 9.97582788e-01 & 2.04723407e-16 & -1.68588444e-03 \\ 1.17597980e-16 & 7.06022786e-03 & 6.82149076e-16 & -5.48032346e-17 & 9.99733713e-01 & -4.84262887e-16 \\ 1.00042919e-02 & -1.05033502e-15 & 3.06771331e-16 & 6.39114790e-03 & -1.42355472e-16 & 1.01500077e+00 \end{pmatrix}$$

Figure A.6: The transition matrices of the extracted WFAs by **RGR**(15) (for $\sigma \in \{(,), 1, 2, 3, 4\}$)

$$A_5 = \begin{pmatrix} 9.99463469e-01 & -5.61082820e-17 & 4.38814586e-17 & 1.22067377e-04 & 6.12978077e-17 & -2.04093919e-04 \\ 1.15318176e-17 & 1.00100869e+00 & -1.21912117e-17 & -4.11424946e-18 & 6.36064247e-04 & -2.67033033e-16 \\ 2.02699532e-19 & -7.22607534e-16 & 1.00029543e+00 & 2.09623345e-16 & -2.30961854e-16 & 3.49224654e-16 \\ -1.93123658e-03 & 2.16064799e-16 & 6.43419914e-16 & 9.98755732e-01 & 2.04693783e-16 & -1.62514201e-03 \\ 1.17159685e-16 & 1.05557406e-03 & 6.82702060e-16 & -5.57875540e-17 & 1.00498703e+00 & -4.86899833e-16 \\ 1.18051058e-02 & -1.04665878e-15 & 3.04945578e-16 & 2.66653121e-03 & -1.49108601e-16 & 1.01340563e+00 \end{pmatrix}$$

$$A_6 = \begin{pmatrix} 9.99270606e-01 & -5.58880977e-17 & 4.41098581e-17 & 5.54905278e-04 & 6.14469389e-17 & -1.46379686e-04 \\ 1.15941865e-17 & 1.00118061e+00 & -1.22843592e-17 & -4.16283361e-18 & 5.63901004e-04 & -2.67024145e-16 \\ 7.64012831e-20 & -7.23401942e-16 & 1.00155801e+00 & 2.09960987e-16 & -2.31395681e-16 & 3.49616662e-16 \\ -6.63387748e-04 & 2.15542630e-16 & 6.42800551e-16 & 9.97574678e-01 & 2.04705718e-16 & -1.44911555e-03 \\ 1.15518342e-16 & -1.02797853e-03 & 6.82862561e-16 & -5.35407556e-17 & 1.00336556e+00 & -4.85213752e-16 \\ 5.57258016e-03 & -1.04098887e-15 & 3.10119657e-16 & 1.04511511e-02 & -1.46075233e-16 & 1.01303930e+00 \end{pmatrix}$$

$$A_7 = \begin{pmatrix} 9.99237402e-01 & -5.58952374e-17 & 4.40907525e-17 & 5.48695123e-04 & 6.14799665e-17 & -9.97701660e-05 \\ 1.15849561e-17 & 1.00108512e+00 & -1.21853831e-17 & -4.18844928e-18 & 6.02383979e-04 & -2.67037089e-16 \\ 2.12971526e-19 & -7.22355438e-16 & 1.00017887e+00 & 2.09645978e-16 & -2.31039310e-16 & 3.49230916e-16 \\ -1.36321195e-03 & 2.15240106e-16 & 6.42861686e-16 & 9.97673235e-01 & 2.04443137e-16 & -1.82208222e-03 \\ 1.17346570e-16 & 4.88068735e-03 & 6.82433269e-16 & -5.49430977e-17 & 1.00282769e+00 & -4.85969900e-16 \\ 1.06710808e-02 & -1.04844637e-15 & 3.06438734e-16 & 6.19596029e-03 & -1.45096416e-16 & 1.01493448e+00 \end{pmatrix}$$

$$A_8 = \begin{pmatrix} 9.99485693e-01 & -5.60463491e-17 & 4.38945978e-17 & 2.23945615e-04 & 6.14354582e-17 & -4.53253526e-05 \\ 1.15874726e-17 & 1.00140298e+00 & -1.21941168e-17 & -4.15369161e-18 & 3.55804959e-04 & -2.66949792e-16 \\ 2.41905849e-19 & -7.22115784e-16 & 9.99859058e-01 & 2.09479615e-16 & -2.30973028e-16 & 3.49093881e-16 \\ -2.07582058e-03 & 2.15463744e-16 & 6.43310294e-16 & 9.98594124e-01 & 2.04913786e-16 & -1.67941050e-03 \\ 1.18327579e-16 & 7.51634097e-03 & 6.82147238e-16 & -5.64154854e-17 & 1.00141082e+00 & -4.85963575e-16 \\ 1.31311788e-02 & -1.05385406e-15 & 3.03514970e-16 & 6.98588331e-04 & -1.45893667e-16 & 1.01363345e+00 \end{pmatrix}$$

$$A_9 = \begin{pmatrix} 9.99315111e-01 & -5.59700858e-17 & 4.39955085e-17 & 3.37854024e-04 & 6.13374084e-17 & -1.83554695e-04 \\ 1.16759138e-17 & 1.00149706e+00 & -1.22979062e-17 & -4.16208693e-18 & 1.23487542e-04 & -2.66766491e-16 \\ 3.37885984e-19 & -7.23262350e-16 & 1.00075052e+00 & 2.09528596e-16 & -2.30850079e-16 & 3.49372665e-16 \\ -1.33702792e-03 & 2.16271556e-16 & 6.43198414e-16 & 9.98592080e-01 & 2.05060444e-16 & -1.03477715e-03 \\ 1.16912096e-16 & -4.78824593e-03 & 6.83072626e-16 & -5.63467680e-17 & 1.00742399e+00 & -4.87380576e-16 \\ 1.37061163e-02 & -1.04360630e-15 & 3.03799727e-16 & -5.37634224e-04 & -1.52428625e-16 & 1.01215221e+00 \end{pmatrix}$$

Figure A.7: The transition matrices of the extracted WFAs by **RGR**(15) (for $\sigma \in \{5, 6, 7, 8, 9\}$)

Bibliography

[1] Marianne Akian, Stéphane Gaubert, and Alexander Guterman. Linear independence over tropical semirings and beyond. *Contemporary Mathematics*, 495:1, 2009.

[2] A. Albarghouthi and K. L. McMillan. Beautiful Interpolants. In *Computer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 2013.

[3] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. In Claire Mathieu, editor, *Proc. SODA 2009*, pages 835–844. SIAM, 2009.

[4] Hirokazu Anai and Pablo A. Parrilo. Convex quantifier elimination for semidefinite programming. In *Proceedings of the International Workshop on Computer Algebra in Scientific Computing, CASC*, 2003.

[5] Sidharta Andalam, Avinash Malik, Partha S. Roop, and Mark L. Trew. Hybrid automata model of the heart for formal verification of pacemakers. In Goran Frehse and Matthias Althoff, editors, *ARCH at CPSWeek 2016, 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems, Vienna, Austria*, volume 43 of *EPiC Series in Computing*, pages 9–17. EasyChair, 2016.

[6] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[7] Stéphane Ayache, Rémi Eyraud, and Noé Goudian. Explaining black boxes on sequential data using weighted automata. In Olgierd Unold, Witold Dyrka, and Wojciech Wieczorek, editors, *Proc. ICGI 2018*, volume 93 of *Proceedings of Machine Learning Research*, pages 81–103. PMLR, 2018.

[8] P. Backeman, P. Rümmer, and A. Zeljic. Bit-Vector Interpolation and Quantifier Elimination by Lazy Reduction. In *Formal Methods in Computer Aided Design*, pages 1–10. IEEE, 2018.

[9] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[10] Thomas Ball, Byron Cook, Vladimir Levin, and Sriram K. Rajamani. SLAM and static driver verifier: Technology transfer of formal methods inside microsoft. In *IFM*, volume 2999 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004.

[11] Borja Balle, Pascale Gourdeau, and Prakash Panangaden. Bisimulation metrics for weighted automata. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proc. ICALP 2017*, volume 80 of *LIPIcs*, pages 103:1–103:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[12] Borja Balle and Mehryar Mohri. Learning weighted automata. In Andreas Maletti, editor, *Proc. CAI 2015*, volume 9270 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2015.

[13] Borja Balle, Prakash Panangaden, and Doina Precup. A canonical form for weighted automata and applications to approximate minimization. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 701–712. IEEE Computer Society, 2015.

[14] C. Barrett, D. Dill, and J. Levitt. A Decision Procedure for Bit-Vector Arithmetic. In *Design and Automation Conference*, pages 522–527, 1998.

[15] Yves Bertot and Pierre Castran. *Interactive Theorem Proving and Program Development: Coq'Art The Calculus of Inductive Constructions*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[16] Frédéric Besson. Fast reflexive arithmetic tactics the linear case and beyond. In Thorsten Altenkirch and Conor McBride, editors, *Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, volume 4502 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2006.

[17] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. The software model checker BLAST. *International Journal on Software Tools for Technology Transfer*, 9(5-6):505–525, 2007.

[18] J. Bochnak, M. Coste, and M.-F. Roy. *Real algebraic geometry*. Springer, 1999.

[19] Max Bramer. *Ensemble Classification*, pages 209–220. Springer London, London, 2013.

[20] R. E. Bryant, D. Kroening, J. Ouaknine, S. A. Seshia, O. Strichman, and B. Brady. Deciding Bit-Vector Arithmetic with Abstraction. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 358–372. Springer, 2007.

[21] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proc. KDD 2006*, pages 535–541, 2006.

[22] Peter Butkovič. *Max-linear systems : theory and algorithms*. Springer monographs in mathematics. Springer, 2010.

[23] Fraser Cameron, Georgios E. Fainekos, David M. Maahs, and Sriram Sankaranarayanan. Towards a verified artificial pancreas: Challenges and solutions for runtime verification. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, volume 9333 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2015.

[24] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010.

[25] Arindam Chaudhuri. *Visual and Text Sentiment Analysis through Hierarchical Deep Learning Networks*. Springer Briefs in Computer Science. Springer, 2019.

[26] Mingshuai Chen, Jian Wang, Jie An, Bohua Zhan, Deepak Kapur, and Naijun Zhan. NIL: learning nonlinear interpolants. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 178–196. Springer, 2019.

[27] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[28] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. The MathSAT5 SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2013.

[29] A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 397–412. Springer, 2008.

[30] A. Cimatti, A. Griggio, and R. Sebastiani. Interpolant Generation for UTVPI. In *CADE*, pages 167–182, 2009.

[31] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model verifier. In Nicolas Halbwachs and Doron A. Peled, editors, *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, volume 1633 of *Lecture Notes in Computer Science*, pages 495–499. Springer, 1999.

[32] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[33] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 1999.

[34] Michael Colón, Sriram Sankaranarayanan, and Henny Sipma. Linear invariant genera-

tion using non-linear constraint solving. In Hunt Jr. and Somenzi [65], pages 420–432.

[35] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In Alfred V. Aho, Stephen N. Zilles, and Thomas G. Szymanski, editors, *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978*, pages 84–96. ACM Press, 1978.

[36] Raymond A. Cuninghame-Green. *Minimax algebra*. Number 166 in Lecture notes in economics and mathematical systems. Springer-Verlag, 1979.

[37] Liyun Dai. The tool `aiSat`. `github.com/djuanbei/aiSat`, cloned on January 17th, 2017.

[38] Liyun Dai, Ting Gan, Bican Xia, and Naijun Zhan. Barrier certificates revisited. *J. Symb. Comput.*, 80:62–86, 2017.

[39] Liyun Dai, Bican Xia, and Naijun Zhan. Generating non-linear interpolants by semidefinite programming. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2013.

[40] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

[41] B. De Schutter and B. De Moor. Matrix factorization and minimal state space realization in the max-plus algebra. In *Proc. ACC 1997 (Cat. No.97CH36041)*, volume 5, pages 3136–3140, 1997.

[42] Bart De Schutter and Bart De Moor. Minimal realization in the max algebra is an extended linear complementarity problem. *Systems & Control Letters*, 25:103–111, 07 2000.

[43] Y. Demyanova, P. Rümmer, and F. Zuleger. Systematic Predicate Abstraction Using Variable Roles. In *NASA Formal Methods*, volume 10227 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2017.

[44] Samuel Drews and Loris D'Antoni. Learning symbolic automata. In Axel Legay and Tiziana Margaria, editors, *Proc. TACAS 2017*, volume 10205 of *LNCS*, pages 173–189, 2017.

[45] M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2009.

[46] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. Deepstellar: model-based quantitative analysis of stateful deep learning systems. In Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo, editors, *Proc. ESEC/FSE 2019*, pages 477–487. ACM, 2019.

[47] M. Elder, J. Lim, T. Sharma, T. Andersen, and T. Reps. Abstract Domains of Affine Relations. *ACM Transactions on Programming Languages and Systems*, 36, 2014.

[48] Michel Fliess. Matrices de hankel. *J. Math. Pures Appl*, 53(9):197–222, 1974.

[49] Ting Gan, Liyun Dai, Bican Xia, Naijun Zhan, Deepak Kapur, and Mingshuai Chen. Interpolant synthesis for quadratic polynomial inequalities and combination with EUF. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, volume 9706 of *Lecture Notes in Computer Science*, pages 195–212. Springer, 2016.

[50] Sicun Gao and Damien Zufferey. Interpolants in nonlinear theories over the reals. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 625–641. Springer, 2016.

[51] S. Graf and H. Saïdi. Construction of Abstract State Graphs with PVS. In *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1997.

[52] A. Griggio. Effective Word-Level Interpolation for Software Verification. In *Formal Methods in Computer-Aided Design*, pages 28–36. IEEE, 2011.

[53] A. Griggio, T. T. H. Le, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Linear Integer Arithmetic. *Logical Methods in Computer Science*, 8(3), 2010.

[54] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proc. ICML 2015*, pages 1737–1746, 2015.

[55] Arie Gurfinkel, Simone Fulvio Rollini, and Natasha Sharygina. Interpolation properties and sat-based model checking. In Dang Van Hung and Mizuhito Ogawa, editors, *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, volume 8172 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2013.

[56] Elena Gutiérrez, Takamasa Okudono, Masaki Waga, and Ichiro Hasuo. Genetic algorithm for the weight maximization problem on weighted automata. In Carlos Artemio Coello Coello, editor, *GECCO '20: Genetic and Evolutionary Computation Conference,*

Cancún Mexico, July 8-12, 2020, pages 699–707. ACM, 2020.

[57] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Proc. NIPS 2015*, pages 1135–1143, 2015.

[58] John Harrison. Verifying nonlinear real formulas via sums of squares. In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics, 20th International Conference, TPHOLs 2007, Kaiserslautern, Germany, September 10-13, 2007, Proceedings*, volume 4732 of *Lecture Notes in Computer Science*, pages 102–118. Springer, 2007.

[59] Léo Henry, Thierry Jéron, and Nicolas Markey. Active learning of timed automata with unobservable resets. In Nathalie Bertrand and Nils Jansen, editors, *Proc. FORMATS 2020*, volume 12288 of *LNCS*, pages 144–160. Springer, 2020.

[60] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy Abstraction. In *Principles of Programming Languages*, pages 58–70, 2002.

[61] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In Jones and Leroy [71], pages 232–244.

[62] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[63] Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Software Eng.*, 23(5):279–295, 1997.

[64] Yosuke Horiuchi. FINAL FANTASY XV POCKET EDITIONを支えるAWSサーバレス技術, 2016. `https://d1.awsstatic.com/events/jp/2018/summit/tokyo/customer/37.pdf`, Accessed in 2020.

[65] Warren A. Hunt Jr. and Fabio Somenzi, editors. *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*. Springer, 2003.

[66] Radoslav Ivanov, Taylor J. Carpenter, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Case study: verifying the safety of an autonomous racing car with a neural network controller. In Aaron D. Ames, Sanjit A. Seshia, and Jyotirmoy Deshmukh, editors, *HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21-24, 2020*, pages 28:1–28:7. ACM, 2020.

[67] H. Jain, E. M. Clarke, and O. Grumberg. Efficient Craig interpolation for linear Diophantine (dis)equations and linear modular equations. *Formal Methods in System Design*, 35(1):6–39, 2009.

[68] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Ryan Gardner, Aurora Schmidt, Erik Zawadzki, and André Platzer. Formal verification of ACAS x, an industrial airborne collision avoidance system. In Alain Girault and Nan Guan, editors, *2015*

International Conference on Embedded Software, EMSOFT 2015, Amsterdam, Netherlands, October 4-9, 2015, pages 127–136. IEEE, 2015.

[69] Ranjit Jhala and Kenneth L. McMillan. Interpolant-based transition relation approximation. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2005.

[70] Ranjit Jhala and Kenneth L. McMillan. A practical and complete approach to predicate refinement. In Holger Hermanns and Jens Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings*, volume 3920 of *Lecture Notes in Computer Science*, pages 459–473. Springer, 2006.

[71] Neil D. Jones and Xavier Leroy, editors. *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*. ACM, 2004.

[72] Erich Kaltofen, Bin Li, Zhengfeng Yang, and Lihong Zhi. Exact certification of global optimality of approximate factorizations via rationalizing sums-of-squares with floating point scalars. In J. Rafael Sendra and Laureano González-Vega, editors, *Symbolic and Algebraic Computation, International Symposium, ISSAC 2008, Linz/Hagenberg, Austria, July 20-23, 2008, Proceedings*, pages 155–164. ACM, 2008.

[73] D. Kapur, R. Majumdar, and C. G. Zarba. Interpolation for Data Structures. In *Foundations of Software Engineering*, pages 105–116, 2006.

[74] M. Karr. Affine Relationships among Variables of a Program. *Acta Informatica*, 6:133–151, 1976.

[75] Masashi Kikuchi. CCS Injection脆弱性(CVE-2014-0224)発見の経緯についての紹介｜株式会社レピダム, 2020. `https://lepidum.co.jp/blog/2014-06-05/CCS-Injection/`.

[76] A. King and H. Søndergaard. Automatic Abstraction for Congruences. In *Verification, Model Checking, and Abstract Interpretation*, volume 9583 of *Lecture Notes in Computer Science*, pages 197–213. Springer, 2010.

[77] Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013.

[78] D. Kroening and G. Weissenbacher. Lifting Propositional Interpolants to the Word-Level. In *Formal Methods in Computer-Aided Design*, pages 85–89. IEEE, 2007.

[79] Daniel Kroening and Michael Tautschnig. CBMC - C bounded model checker - (competition contribution). In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms*

for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings, volume 8413 of *Lecture Notes in Computer Science*, pages 389–391. Springer, 2014.

[80] Stefan Kupferschmid and Bernd Becker. Craig interpolation in the presence of non-linear constraints. In Uli Fahrenberg and Stavros Tripakis, editors, *Formal Modeling and Analysis of Timed Systems - 9th International Conference, FORMATS 2011, Aalborg, Denmark, September 21-23, 2011. Proceedings*, volume 6919 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011.

[81] Markus Alexander Kuppe, Leslie Lamport, and Daniel Ricketts. The TLA+ toolbox. In Rosemary Monahan, Virgile Prevosto, and José Proença, editors, *Proceedings Fifth Workshop on Formal Integrated Development Environment, F-IDE at FM 2019, Porto, Portugal, 7th October 2019*, volume 310 of *EPTCS*, pages 50–62, 2019.

[82] S. Lang. *Introduction to Diophantine Approximations*. Springer books on elementary mathematics. Springer, 1995.

[83] Xavier Leroy, Sandrine Blazy, Daniel Kästner, Bernhard Schommer, Markus Pister, and Christian Ferdinand. Compcert – a formally verified optimizing compiler. In *ERTS 2016: Embedded Real Time Software and Systems*. SEE, 2016.

[84] Yuding Liang and Kenny Qili Zhu. Automatic generation of text descriptive comments for code blocks. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 5229–5236. AAAI Press, 2018.

[85] Wang Lin, Min Wu, Zhengfeng Yang, and Zhenbing Zeng. Proving total correctness and generating preconditions for loop programs via symbolic-numeric computation methods. *Frontiers of Computer Science*, 8(2):192–202, 2014.

[86] ShaoHua Lv, Jian Wang, YinQi Yang, and Jiqiang Liu. Intrusion prediction with system-call sequence-to-sequence model. *IEEE Access*, 6:71413–71421, 2018.

[87] Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995.

[88] K. McMillan. An Interpolating Theorem Prover. *Theoretical Computer Science*, 345(1):101–121, 2005.

[89] K. L. McMillan. Lazy Abstraction with Interpolants. In *Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2006.

[90] Kenneth L. McMillan. Interpolation and sat-based model checking. In Hunt Jr. and

Somenzi [65], pages 1–13.

[91] Kenneth L. McMillan. Applications of craig interpolants in model checking. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3440 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[92] Kenneth L. McMillan. Interpolation and Model Checking. In *Handbook of Model Checking*, pages 421–446. Springer, 2018.

[93] Joshua J. Michalenko, Ameesh Shah, Abhinav Verma, Richard G. Baraniuk, Swarat Chaudhuri, and Ankit B. Patel. Representing formal languages: A comparison between finite automata and recurrent neural networks. In *Proc. ICLR 2019*. OpenReview.net, 2019.

[94] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[95] Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski. Learning nominal automata. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proc. POPL 2017*, pages 613–625. ACM, 2017.

[96] M. Möller and H. Rue. Solving Bit-Vector Equations. In *Formal Methods in Computer-Aided Design*, volume 1522 of *Lecture Notes in Computer Science*, pages 36–48, 1998.

[97] M. Müller-Olm and H. Seidl. Analysis of Modular Arithmetic. *ACM Transactions on Programming Languages and Systems*, 29(5):29, 2007.

[98] Chris Newcombe. Why amazon chose TLA +. In Yamine Aït Ameur and Klaus-Dieter Schewe, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings*, volume 8477 of *Lecture Notes in Computer Science*, pages 25–39. Springer, 2014.

[99] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. Use of Formal Methods at Amazon Web Services, 2014. `https://lamport.azurewebsites.net/tla/formal-methods-amazon.pdf`, Accessed in 2020.

[100] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.

[101] Takamasa Okudono and Andy King. Mind the gap: Bit-vector interpolation recast over linear integer arithmetic. In Armin Biere and David Parker, editors, *Tools and Algo-*

rithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I, volume 12078 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2020.

[102] Takamasa Okudono, Yuki Nishida, Kensuke Kojima, Kohei Suenaga, Kengo Kido, and Ichiro Hasuo. Sharper and simpler nonlinear interpolants for program verification. In Bor-Yuh Evan Chang, editor, *Programming Languages and Systems - 15th Asian Symposium, APLAS 2017, Suzhou, China, November 27-29, 2017, Proceedings*, volume 10695 of *Lecture Notes in Computer Science*, pages 491–513. Springer, 2017.

[103] Takamasa Okudono, Masaki Waga, Taro Sekiyama, and Ichiro Hasuo. Weighted automata extraction from recurrent neural networks via regression on state spaces. In *Proc. AAAI 2020*, pages 5306–5314. AAAI Press, 2020.

[104] Christian W. Omlin and C. Lee Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.

[105] Nicolas Papernot, Patrick D. McDaniel, Ananthram Swami, and Richard E. Harang. Crafting adversarial input sequences for recurrent neural networks. In Jerry Brand, Matthew C. Valenti, Akinwale Akinpelu, Bharat T. Doshi, and Bonnie L. Gorsic, editors, *Proc. MILCOM 2016*, pages 49–54. IEEE, 2016.

[106] P.A. Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Inst. of Tech., 2000.

[107] Pablo A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96(2):293–320, 2003.

[108] Lawrence C. Paulson. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In Renate A. Schmidt, Stephan Schulz, and Boris Konev, editors, *Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning, PAAR-2010, Edinburgh, Scotland, UK, July 14, 2010*, volume 9 of *EPiC Series in Computing*, pages 1–10. EasyChair, 2010.

[109] Helfried Peyrl and Pablo A. Parrilo. Computing sum of squares decompositions with rational coefficients. *Theor. Comput. Sci.*, 409(2):269–281, 2008.

[110] André Platzer, Jan-David Quesel, and Philipp Rümmer. Real world verification. In Renate A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 485–501. Springer, 2009.

[111] A. Pnueli, Y. Rodeh, O. Strichman, and M. Siegel. The Small Model Property: How small can it be? *Information and Computation*, 178(1):279–293, 2002.

[112] M Putinar. Positive polynomials on compact semi-algebraic sets. *Indiana Univ. Math.*

Journ., 42(3):969–984, 1993.

[113] Guillaume Rabusseau, Tianyu Li, and Doina Precup. Connecting weighted automata and recurrent neural networks through spectral learning. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proc. AISTATS 2019*, volume 89 of *Proceedings of Machine Learning Research*, pages 1630–1639. PMLR, 2019.

[114] Pierre Roux, Yuen-Lam Voronin, and Sriram Sankaranarayanan. Validating numerical semidefinite programming solvers for polynomial invariants. In Xavier Rival, editor, *Static Analysis - 23rd International Symposium, SAS 2016, Edinburgh, UK, September 8-10, 2016, Proceedings*, volume 9837 of *Lecture Notes in Computer Science*, pages 424–446. Springer, 2016.

[115] S.M. Rump. Verification of positive definiteness. *BIT Numerical Mathematics*, 46(2):433–452, 2006.

[116] Andrey Rybalchenko and Viorica Sofronie-Stokkermans. Constraint solving for interpolation. In Byron Cook and Andreas Podelski, editors, *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings*, volume 4349 of *Lecture Notes in Computer Science*, pages 346–362. Springer, 2007.

[117] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Non-linear loop invariant generation using gröbner bases. In Jones and Leroy [71], pages 318–329.

[118] Stefano Schivo and Rom Langerak. Discretization of continuous dynamical systems using UPPAAL. In Joost-Pieter Katoen, Rom Langerak, and Arend Rensink, editors, *ModelEd, TestEd, TrustEd - Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday*, volume 10500 of *Lecture Notes in Computer Science*, pages 297–315. Springer, 2017.

[119] Roy Schwartz, Sam Thomson, and Noah A. Smith. Bridging CNNs, RNNs, and weighted finite-state machines. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 295–305, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[120] Amazon Web Services. Amazon EC2 FAQs, 2019. `https://aws.amazon.com/ec2/faqs/`, accessed in 2019.

[121] Rahul Sharma, Saurabh Gupta, Bharath Hariharan, Alex Aiken, Percy Liang, and Aditya V. Nori. A data driven approach for algebraic loop invariants. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7792 of *Lecture Notes in Computer Science*, pages

574–592. Springer, 2013.

[122] Zhen Shen, Wenzheng Bao, and De-Shuang Huang. Recurrent neural network for predicting transcription factor binding sites. *Scientific reports*, 8(1):15270, 2018.

[123] A. Simon and A. King. Taming the Wrapping of Integer Arithmetic. In *Static Analysis Symposium*, volume 4634 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 2007.

[124] Gilbert Stengle. A Nullstellensatz and a Positivstellensatz in semialgebraic geometry. *Mathematische Annalen*, 207(2):87–97, 1974.

[125] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Proc. NIPS 2014*, pages 3104–3112, 2014.

[126] Tachio Terauchi. Explaining the effectiveness of small refinement heuristics in program verification with CEGAR. In Sandrine Blazy and Thomas Jensen, editors, *Static Analysis - 22nd International Symposium, SAS 2015, Saint-Malo, France, September 9-11, 2015, Proceedings*, volume 9291 of *Lecture Notes in Computer Science*, pages 128–144. Springer, 2015.

[127] K. C. Toh, M.J. Todd, and R. H. Tütüncü. Sdpt3 – a matlab software package for semidefinite programming. *OPTIMIZATION METHODS AND SOFTWARE*, 11:545–581, 1999.

[128] L.W. Tu. *An Introduction to Manifolds*. Universitext. Springer New York, 2007.

[129] Gerco van Heerdt, Clemens Kupke, Jurriaan Rot, and Alexandra Silva. Learning weighted automata over principal ideal domains. In Jean Goubault-Larrecq and Barbara König, editors, *Proc. FoSSaCS*, volume 12077 of *LNCS*, pages 602–621. Springer, 2020.

[130] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In Faron Moller and Graham Birtwistle, editors, *Logics for Concurrency*, volume 1043 of *Lect. Notes Comp. Sci.*, pages 238–266. Springer Berlin Heidelberg, 1996.

[131] Willem Visser, Klaus Havelund, Guillaume P. Brat, Seungjoon Park, and Flavio Lerda. Model checking programs. *Autom. Softw. Eng.*, 10(2):203–232, 2003.

[132] Hayato Waki and Masakazu Muramatsu. A facial reduction algorithm for finding sparse SOS representations. *Oper. Res. Lett.*, 38(5):361–365, 2010.

[133] Cheng Wang and Mathias Niepert. State-regularized recurrent neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6596–6606. PMLR, 2019.

[134] Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural

networks using queries and counterexamples. In Jennifer G. Dy and Andreas Krause, editors, *Proc. ICML 2018*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 5244–5253. JMLR.org, 2018.

[135] Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision rnns for language recognition. In Iryna Gurevych and Yusuke Miyao, editors, *Proc. ACL 2018, Volume 2: Short Papers*, pages 740–745. Association for Computational Linguistics, 2018.

[136] Xi Xiao, Shaofeng Zhang, Francesco Mercaldo, Guangwu Hu, and Arun Kumar Sangaiah. Android malware detection based on system call sequences and LSTM. *Multimedia Tools Appl.*, 78(4):3979–3999, 2019.

[137] Tomoya Yamaguchi, Martin Brain, Chirs Ryder, Yosikazu Imai, and Yoshiumi Kawamura. Application of abstract interpretation to the automotive electronic control system. In Constantin Enea and Ruzica Piskac, editors, *Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-15, 2019, Proceedings*, volume 11388 of *Lecture Notes in Computer Science*, pages 425–445. Springer, 2019.

[138] Tomoya Yamaguchi, Tomoyuki Kaga, Alexandre Donzé, and Sanjit A. Seshia. Combining requirement mining, software model checking and simulation-based verification for industrial automotive systems. In Ruzica Piskac and Muralidhar Talupur, editors, *2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016*, pages 201–204. IEEE, 2016.

[139] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proc. ACL 1995*, pages 189–196, 1995.

[140] Geoffrey Zweig, Chengzhu Yu, Jasha Droppo, and Andreas Stolcke. Advances in all-neural speech recognition. In *Proc. ICASSP 2017*, pages 4805–4809. IEEE, 2017.