UNIVERSITY OF
CAMBRIDGE

Descriptive complexity of constraint problems

Pengming Wang

Gonville and Caius College

This dissertation is submitted on 27 July, 2017 for the degree of Doctor of Philosophy

Abstract

Constraint problems are a powerful framework in which many common combinatorial problems can be expressed. Examples include graph colouring problems, Boolean satisfaction, graph cut problems, systems of equations, and many more. One typically distinguishes between constraint satisfaction problems (CSPs), which model strictly decision problems, and so-called valued constraint satisfaction problems (VCSPs), which also include optimisation problems.

A key open problem in this field is the long-standing dichotomy conjecture by Feder and Vardi. It claims that CSPs only fall into two categories: Those that are NP-complete, and those that are solvable in polynomial time. This stands in contrast to Ladner's theorem, which, assuming P $\neq$ NP, guarantees the existence of problems that are neither NP-complete, nor in P, making CSPs an exceptional class of problems. While the Feder-Vardi conjecture is proven to be true in a number of special cases, it is still open in the general setting. (Recent claims affirming the conjecture are not considered here, as they have not been peer-reviewed yet.)

In this thesis, we approach the complexity of constraint problems from a descriptive complexity perspective. Namely, instead of studying the computational resources necessary to solve certain constraint problems, we consider the expressive power necessary to define these problems in a logic. We obtain several results in this direction. For instance, we show that Schaefer's dichotomy result for the case of CSPs over the Boolean domain can be framed as a definability result: Either a CSP is definable in fixed-point logic with rank (FPR), or it is NP-hard. Furthermore, we show that a dichotomy exists also in the general case. For VCSPs over arbitrary domains, we show that a VCSP is either definable in fixed-point logic with counting (FPC), or it is not definable in infinitary logic with counting ($\mathcal{C}^\omega$).

We show that these definability dichotomies also have algorithmic implications. In particular, using our results on the definability of VCSPs, we prove a dichotomy on the number of levels in the Lasserre hierarchy necessary to obtain an exact solution: For a finite-valued VCSP, either it is solved by the first level of the hierarchy, or one needs $\Omega(n)$ levels.

Finally, we explore how other methods from finite model theory can be useful in the context of constraint problems. We consider pebble games for finite variable logics in this context, and expose new connections between CSPs, pebble games, and homomorphism preservation results.

Declaration

This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except where specified in the text. This dissertation is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other university. This dissertation does not exceed the prescribed limit of 60 000 words.

Pengming Wang
July, 2017

Acknowledgements

I feel truly humbled to have spent my time as a doctoral student under the supervision of Anuj Dawar. His insights sparked many ideas in this thesis, and without his steady guidance and support the present thesis would not have been possible. I am deeply grateful for this.

Big thanks goes to Albert Atserias and Thomas Sauerwald for their time and diligence reading this thesis. Their comments and suggestions substantially improved this work, and is greatly appreciated.

Many thanks also to Samson Abramsky for encouraging our collaboration, and many fruitful discussions.

I feel privileged to have been part of the Computer Laboratory. I want to thank the many colleagues and friends who made my time here enjoyable and fulfilling. In particular, I'd like to thank Jannis Bulian and Gregory Wilsenach for their companionship and many enjoyable conversations.

Outside the Lab, I'm thankful to Anton Krohmer, Johannes Birkenheier, and Hendrik Molter for the many adventures we've had over the last years, be it climbing the Hearthstone ladder, or some mountain in the Alps. It is always a joy to join you guys after a long day.

I am also deeply grateful to my wife Zongyan Huang for all the encouragement and support she gave me along this journey. With you even the most stressful days turn colourful and bright.

Finally, I would like to thank my parents for their infinite love and support. Through your support, home always felt close.

Contents

Chapter 1

Introduction

Constraint problems describe a framework of search problems where the solution space is restricted by a given set of explicit constraints. Many well-studied algorithmic tasks, such as determining satisfiability of Boolean formulas, solving systems of equations, or finding a minimum vertex cover in a graph, are essentially constraint problems. Moreover, instances of constraint problems also commonly appear in our practical life.

Example 1.1. Imagine you are tasked with designing a seating plan for your wedding party. You have arranged for enough seating for all of your guests, and the seats are grouped into several tables. Your task is to assign each guest a seat. However, you are also aware of certain individual seating preferences: Some guests requested to be seated at the same table with some other person, while some others would like to avoid sitting at the same table with someone else. Finding an assignment of seats that respects these preferences (if it exists at all) is a constraint problem.

In the above example, we can imagine different ways to formulate the task of the seating planner. For instance, we could ask whether it is even possible to find an arrangment that accomodates all seating preferences. This is a decision problem and falls under the framework of so-called *constraint satisfaction problems* (CSPs). On the other hand, if it is not possible after all, we would still be interested in finding a seating plan that satisfies as many preferences as possible. Problems of this kind are called *constraint maximisation problems*, and are part of the larger framework of *constraint optimisation*, also commonly called *valued constraint satisfaction problems* (VCSPs).

More formally, a constraint satisfaction problem consists of a set of *variables* (the guests), a *domain* of values that can be assigned to the variables (the seats), and a set of *constraints* in form of relations over subsets of the variables (relations encoding "A prefers B" and "A avoids B"). The case of constraint optimisation problems is modelled by assigning number values to different assignments of constraints, and then maximising or minimising the sum of the values over all constraints. For the seating plan example this means that we could assign a value of 1 to each satisfied constraint, and 0 to each unsatisfied one, with the objective to find an assignment

maximising the sum of these values. By varying the value assignments of each constraint we can express many optimisation problems in this way.

Example 1.2. A common problem in network analysis is to determine the *maximum flow*, or equivalently, the *minimum cut* between two endpoints in a network. An instance of this problem is given by a network $G = (V, E)$, a capacity function $c : E \to \mathbb{Q}$, and two distinct vertices $s, t \in V$. For simplicity, let us assume that all capacities are 1. The minimum cut problem asks for a bipartition of the vertices V such that s and t are in different parts, and the number of edges between the two parts is minimised.

We can also view this as a constraint optimisation problem. Take V as the set of variables, and $\{0, 1\}$ as the domain. Each edge $e = (u, v)$ defines a constraint on (u, v) that evaluates assignments according to the mapping

$$f(a, b) = \begin{cases} 0 & \text{if } a = b, \\ 1 & \text{otherwise.} \end{cases}$$

For instance, for an assignment that maps $u \mapsto 0$ and $v \mapsto 1$, the constraint on $e = (u, v)$ would yield a value of 1. A minimum cut can then be determined by finding an assignment that minimises the sum of these values over all constraints.

Formally we parameterise constraint problems by a *constraint language* Γ. A constraint language contains the relations used to express the constraints of an instance. To illustrate, in Example 1.1, we would have a constraint language with two binary relations $\Gamma = \{\text{PREFERS}, \text{AVOIDS}\}$, where $(a, b) \in \text{PREFERS}$ if and only if $a = b$, and $(a, b) \in \text{AVOIDS}$ if and only if $a \neq b$. A constraint in the seating plan problem in the form of "X prefers Y" is then satisfied by a seating assignment $X \mapsto a$ and $Y \mapsto b$ only if $a = b$. We can then write $\text{CSP}(\Gamma)$ to mean the class of all seating problem instances. For optimisation problems, constraint languages contain functions instead of relations. In Example 1.2, the constraint language would contain the function f.

The framework of constraint problems unites a wide range of combinatorial decision and optimisation problems under a single formalism. It has found applications in many disciplines, including database theory [91, 74], artificial intelligence [43, 17], graph theory [62], and even physics [46]. From a more theoretical point of view, constraint problems also plays a central role in the field of complexity theory. Due to its generality, constraint problems include many algorithmically tractable problems (e.g. systems of linear equations, maximum flow), but also include various famously difficult problems (e.g. graph k-colouring, Boolean satisfiability), for which no polynomial-time algorithm is known. Determining exactly which properties of a constraint problem imply tractability or hardness has been an ongoing quest in the past decades.

In this thesis we follow this line of research, however using a different measure of complexity. Classically, computational complexity is measured in terms of the

resources needed to solve a problem, such as the required amount of memory or computing time. In contrast, we are primarily concerned about the *definability* of a problem. That is, instead of asking which resources are needed to *decide* that an input satisfies some property, we measure complexity in terms of the syntactical richness required in a logic to *define* that property. This notion is commonly called *descriptive complexity*.

The main theme of this thesis is to examine constraint problems under the lens of descriptive complexity. In this context, we present a number of positive as well as negative results regarding the definability of constraint problems in certain logics. Moreover, we show that these results also have algorithmic consequences. In particular, we are able to prove a lower bound on the number of iterations needed in the relaxation hierarchy of Lasserre [82] to solve certain constraint optimisation problems exactly. Finally, we also explore other connections between CSPs and logic, and show how certain definability results in the CSP world can be interpreted as a type of preservation theorems in finite model theory. We discuss the contributions of the thesis in more detail in Section 1.2.

1.1 Related work

In both fields of constraint problem research and descriptive complexity, we can look back on a vast body of research in the last decades. For a structured introduction into these areas we recommend the textbooks [88, 68, 83], and recent surveys [79, 8]. Here, we summarise some classic and recent lines of work that are related to the contributions of this thesis.

Complexity of constraint problems. Complexity dichotomies for constraint problems have been a central topic of study since the famous dichotomy conjecture was posed by Feder and Vardi [48]. It conjectures that for any constraint language Γ, CSP(Γ) is either solvable in polynomial time; or it is NP-hard. Since Ladner's theorem [80] states that if P $\neq$ NP, there are problems in NP that are neither in P, nor NP-complete, confirming the conjecture would make CSPs a large, natural class that excludes problems of intermediate hardness.

So far, several restricted cases of the conjecture have been confirmed. Before Feder and Vardi's conjecture, two subclasses of CSPs were known to follow a complexity dichotomy: Schaefer [92] gave a complete classfication of CPSs with Boolean domains, and Hell and Nešetřil [61] showed the dichotomy for constraint languages consisting of a single binary relation. Since then, the conjecture has been proved for a number of additional cases, including CSPs on larger domains [20], CSPs with so-called conservative constraint languages [21, 7, 22], and CSPs over certain classes of directed graphs [11]. Very recently, several authors [23, 102, 87] independently claimed to have resolved the dichotomy conjecture. However, at this time, these works have not been thoroughly peer-reviewed yet, and for the purposes of this thesis, we consider the conjecture still open. In the optimisation framework,

a dichotomy is known to hold for finite-valued CSPs [96], as well as conservative VCSPs [77]. More generally, it has been shown that a positive resolution of the CSP dichotomy conjecture would also confirm the existence of a complexity dichotomy for general VCSPs [76].

Related to the quest of resolving the complexity dichotomy conjecture for CSPs is the task of characterising tractable subclasses of CSPs, and finding suitable algorithms. Interestingly, all known tractable cases of CSPs are essentially solved by two main types of algorithms, namely *local consistency* algorithms [50], and certain algebraic algorithms resembling Gaussian elimination, called *few subpowers* algorithms [33, 24, 66, 15]. The set of CSPs that are tractable via local consistency methods are commonly known as *bounded width* CSPs, while the few subpower algorithms are applicable for constraint languages that satisfy certain algebraic properties. In the optimisation case, *linear programming* has turned out to be a key algorithmic framework to solve tractable VCSPs. The power and limits of linear and semidefinite programs in the context of VCSPs has been studied extensively in [98, 97]. In particular, it has been shown that the applicability of linear programming to VCSPs is closely related to the bounded width characterisation in the CSP case. Moreover, in the special case of finite-valued CSPs where a complexity dichotomy is known, it turns out that the tractable cases are exactly those that are solvable by linear programming [96], while all others are NP-complete.

Significant in the recent progress in this area was the development of an underlying algebraic theory. In this context, each constraint language Γ gives rise to an algebra $\mathrm{Alg}(\Gamma)$, whose element set is the domain, and its operations are the *polymorphisms* of Γ. It turns out that the tractability of $\mathrm{CSP}(\Gamma)$ is fully determined by the set of operations in $\mathrm{Alg}(\Gamma)$ [25]. Choosing algebras as the main objects of study renders CSPs accessible to techniques from universal algebra, which has been the catalyst to many recent complexity results. We will see throughout this thesis that this remains true in many ways when considering definability instead of computational complexity. Early results linking the complexity of $\mathrm{CSP}(\Gamma)$ to algebraic properties of $\mathrm{Alg}(\Gamma)$ include [69, 25, 24]. A key development was the idea to characterise $\mathrm{Alg}(\Gamma)$ in terms of Hobby and McKenzie's tame congruence theory of finite algebras [25, 64], which has been a crucial component in many of the previously mentioned results. Many of the core techniques in the study of CSPs can be adapted to the world of VCSPs [31, 78, 76, 32], which has also led to a number of interreducibility results for CSPs and VCSPs [31, 51].

The descriptive complexity of CSPs has been comparatively less studied. Still, there exist a number of results relating CSPs and definability. In their seminal paper, Feder and Vardi [48] point out connections between CSPs and the logics Datalog and MMSNP, and establish that CSPs that are sovable by local consistency algorithms are expressible in Datalog. This connection between CSPs and Datalog were further deepened by a series of results [35, 34, 75, 45], showing that certain syntactical restrictions of Datalog capture CSP classes that lie in NL or even logspace. In a first negative result, Atserias, Bulatov, and Dawar [5] have shown that $\mathrm{CSP}(\Gamma)$ that

satisfy a certain algebraic property are not definable in infinitary logic with counting ($\mathcal{C}^\omega$). This algebraic property was later shown to characterise exactly the class of bounded width CSPs [81, 10, 9], establishing a definability dichotomy between CSPs definable in Datalog, and those not definable in $\mathcal{C}^\omega$.

Linear and semidefinite optimisation. Due to the existence of efficient solvers [70, 18], linear and semidefinite programs have become popular tools to approach many combinatorial optimisation problems, and are even useful for NP-hard problems. Relaxation hierarchies, such as those of Lovasz-Shrijver [84], Sherali-Adams [94], or Lasserre [82] provide systematic ways to obtain sequences of linear or semidefinite programs that give increasingly accurate approximation schemes for many hard problems. There exists many studies on the power and limitations of these relaxation hierarchies, e.g. [99, 42, 6]. In particular for CSPs, Grigoriev [54] and Schoenebeck [93] established linear lower bounds for the number of levels in the Lasserre hierarchy required to solve a variety of constraint problems exactly. Generalising these results, Thapper and Živný [97] showed that for general VCSPs, the required levels form a dichotomy: Either a VCSP(Γ) is solved by the third level of the Lasserre hierarchy, or it requires linearly many levels. The study of semidefinite relaxation hierarchies also has deep ties into the *unique games conjecture*, which have been surveyed in [71].

Optimisation problems have been studied from the viewpoint of descriptive complexity on a few occasions [72, 85]. A result that is crucial to this thesis is due to Anderson, Dawar, and Holm [3, 4], who show that solutions to linear programs can be defined in fixed-point logic with counting. One of the contributions of this thesis is to show that this can be extended to semidefinite programs as well, subject to certain preconditions.

Finite model theory. Since Fagin's seminal result [47, 68] that a class of finite structures is decidable in NP if and only if it is definable in existential second-order logic, the question of whether a similar characterisation of P exists has been central to the field of descriptive complexity theory [27, 60]. As one of the first candidate logics, extensions of first-order logic with fixed-point operators have attained a prominent role. Independently proven by Immerman [67] and Vardi [100], it is known that inflationary fixed-point logic (IFP) can define exactly all P decidable properties over *ordered* finite structures. However, in the general case, where structures do not come with an order relation, IFP has obvious weaknesses. In particular, it is unable to define many properties that relate to cardinality. For this reason, the extension of IFP with a counting operator, so-called fixed-point logic with counting (FPC) was considered. It turns out FPC is still not the right logic for P, as was shown first by Cai, Fürer, and Immerman [26]. As the known cases of polynomial-time decidable properties that are not definable in FPC all relate to solvability of linear equation systems [5], it is open whether extending FPC with a mechanism to define solvability would suffice to capture P. In this context, fixed-point logic with rank (FPR) has been considered [38, 65]. This logic extends inflationary fixed-point

logic with an operator that expresses the rank of definable matrices. While FPR addresses the known counterexamples of tractable but undefinable problems in FPC, its expressive power is still not well understood, although first results in this area have been found [37, 53].

While FPC is insufficient to define all P properties over arbitrary finite structures, it has been shown to be a robust and powerful logic in its own right. In particular, it is known through a series of results that FPC can express all polynomial-time properties on planar graphs [55], structures of bounded tree-width [57], and more generally all classes of graphs with excluded minors [56].

Pebble games are a well-studied concept in finite model theory, and are commonly used as a tool to characterise the expressive power of a logic. Due to their combinatorial nature, pebble games are also useful to connect logic with algorithmic problems, and played a substantial part in establishing connections between logic and CSPs [75, 14] and graph isomorphism [13]. Pebble games were also instrumental in proving preservation theorems in the finite, such as Rossman's homomorphism preservation theorem [89] for first-order logic, and the preservation theorems by Feder and Vardi in [49].

1.2 Outline and contributions

One of our main goals is to demonstrate a number of applications of techniques from finite model theory to the study of constraint problems. In the following we lay out the organisation of this thesis and list our key results. The contributions in Chapters 3, 4, and 5 are based on joint work with Anuj Dawar, and contains material published in [40, 41]; Chapter 6 is based on joint work with Anuj Dawar and Samson Abramsky, and contains material published in [1].

In Chapter 2 we give a broad overview of the topics treated in the following chapters. This includes an introduction into the basic terminology and concepts of constraint problems, linear and semidefinite optimisation, and logic. Moreover, we give a summary of the relevant results in the field that are used in the remainder of the thesis.

In Chapter 3 we analyse the definability of Boolean CSPs and establish a descriptive complexity version of Schaefer's dichotomy theorem [92]. Namely, we show that for Boolean constraint languages Γ, CSP(Γ) is tractable if and only if it is definable in fixed-point logic with rank (FPR). This result is obtained by inspecting the tractable cases listed by Schaefer, and noting that they are either known to be expressible in Datalog, or can be encoded as systems of linear equations over the Boolean field, which are expressible in FPR [38].

In Chapter 4 we examine the definability of constraint problems over the general domain, and again establish a dichotomy. We show that in the general case of VCSPs, problems fall into one of two cases: For any Γ, either VCSP(Γ) is definable

in fixed-point logic with counting (FPC); or it is not definable in the more powerful infinitary counting logic ($\mathcal{C}^\omega$). Our result relies on the complexity dichotomy result for finite-valued CSPs by Thapper and Živný [96], as well as the definability dichotomy for relational CSPs [81, 5, 10], and provides a definability perspective on the general-valued case. We also use number of technical results from [78] to lift results from the world of CSPs to the more general case of VCSPs.

In Chapter 5 we show that the optimal value of explicitly given semidefinite programs (SDPs) are definable by means of an interpretation in the logic FPC, under a non-emptiness condition. This extends work on a similar result for linear programs by Anderson et al. [3]. As an application, we establish another dichotomy for constraint problems, this time in terms of the number of relaxation levels in the Lasserre hierarchy. Namely we show that in the framework of finite-valued CSPs, either a problem is solved by the first level of the hierarchy; or it is only solved after a linear number of levels in the Lasserre heirarchy. This result extends on some previously known Lasserre lower bounds from [93]. We remark that independently form our work, and using very different techniques, Thapper and Živný recently showed a similar result in [97] for the case of general VCSPs.

In Chapter 6 we consider a common concept in model theory, namely pebble games. These games are usually associated with a logic and are a useful tool to prove inexpressibility of properties in the logic. Motivated by the connection between bounded width CSPs and the pebble game for k-variable existential positive logic [75, 49], we introduce a new formalism to study these types of games. As an application, we present a generic way to produce constructions similar to those of Cai, Fürer, and Immerman [26] that serve as examples of non-isomorphic structures that are indistinguishable in the logic FPC. In this context, we observe that the definability dichotomy for CSPs by Atserias et al. [5] can be seen as a kind of preservation theorem and draw parallels to Rossman's homomorphism preservation theorem for finite structures [89].

Finally, in Chapter 7 we summarise the main results of the thesis, and provide an overview of potential directions for further research.

Chapter 2

Preliminaries and definitions

We start with a brief introduction to the basic concepts of constraint problems, linear and semidefinite optimisation, and logic. Most of the definitions here are standard and can be found in textbooks or surveys, such as [59, 88, 44, 68, 8, 79].

2.1 Common acronyms

CSP	–	Constraint satisfaction problem
VCSP	–	Valued constraint satisfaction problem
LP	–	Linear program
SDP	–	Semidefinite program
BLP	–	Basic linear program
FO	–	First-order logic
IFP	–	Inflationary fixed-point logic
FPC	–	Fixed-point logic with counting
FPR	–	Fixed-point logic with rank

2.2 Notation and conventions

We write $\mathbb{N}$ for the natural numbers, $\mathbb{Z}$ for the integers, $\mathbb{Q}$ for the rational numbers, and use a superscript plus for the non-negative subset, e.g. $\mathbb{Z}^+ = \mathbb{N} \cup \{0\}$. We also write $\mathbb{Q}_\infty$ to denote the set $\mathbb{Q} \cup \{\infty\}$. For a number $d \in \mathbb{N}$, we denote by $[d]$ the initial segment of the natural numbers $\{1, \ldots, d\}$. For a set S, we denote its *power set* by $\wp(S)$, and we write $\wp_k(S)$ for the set of subsets of S that have size at most k. We write $\mathrm{ar}(t)$ to denote the *arity* of t, where t could be a tuple, function, or relation. We use the symbol $\dot\cup$ for the *disjoint union* of sets.

An m-ary *relation* R over a set D is simply a subset of D^m. We sometimes also view R as a function $R : D^m \to \{0, \infty\}$, where $R(x) = 0$ if x is in the relation, and $R(x) = \infty$ otherwise. For the sake of convenience we will use both representations interchangeably, depending on the context.

For our purposes, it is often useful to also consider vectors and matrices as functions. Given sets A and I, an A-valued *vector* v indexed by I is a function $v : I \to A$. Often we simply use the subscript notation, writing v_i for $v(i)$. When there is no explicit index set given, vectors are indexed by $[d]$. A *matrix* M is a vector which is indexed by a product set $I \times J$. We use $M_{i,j}$ to denote $M(i, j)$. Throughout this thesis, vectors and matrices are usually rational-valued, i.e. they are elements of $\mathbb{Q}^I$ or $\mathbb{Q}^{I \times J}$.

For a rational valued vector $v \in \mathbb{Q}^I$, its *norm* $\|v\|$ is defined as the L^2-norm over $\mathbb{Q}^I$. The *inner product* of two vectors $a, b \in \mathbb{Q}^I$ is defined as $\langle a, b\rangle := \sum_{i \in I} a_i b_i$. In the case of matrices, this definition of $\langle \cdot, \cdot \rangle$ coincides with the trace inner product of matrices. The norm of a matrix M is defined as the norm of M seen as a vector. For a set $\mathcal{F} \subseteq \mathbb{Q}^I$ and a vector $v \in \mathbb{Q}^I$, we define the *distance* $\delta(v, \mathcal{F})$ in the usual way, i.e. as $\delta(v, \mathcal{F}) := \min_{x \in \mathcal{F}} \|x - v\|$. The *ball* $\mathcal{B}(v, r)$ around v with radius $r \in \mathbb{Q}$ is then the set $\mathcal{B}(v, r) := \{x \in \mathbb{Q}^I \mid \|x - v\| \leq r\}$.

2.3 Constraint problems

Constraint problems are a powerful framework to express many common combinatorial and optimisation problems, including Boolean satisfiability, graph colouring, transport problems, and more. Classically, one considers *decision* problems in the form of *constraint satisfaction problems* (CSPs), where one is interested in whether a set of constraints can all be satisfied at the same time. If we are however rather interested in *optimising* some assigned value to the constraints, we are in the framework of *constraint optimisation problems*, or *valued CSPs* (VCSPs).

In this thesis we study the descriptive complexity of various constraint problems of both types. More specifically, we consider the complexity of constraint problems parameterised by a fixed *domain*, and a *constraint language*.

Definition 2.1. A *domain* D is a finite, non-empty set. A *constraint language* Γ over D is a set of functions over D, where each $f \in \Gamma$ has an associated arity $m = \mathrm{ar}(f)$, and $f : D^m \to \mathbb{Q}_\infty$.

We say Γ is *relational*, if Γ only contains relations.

Definition 2.2. Let Γ be a relational constraint language. An instance of the *constraint satisfaction problem* over Γ, or CSP(Γ), is a pair $I = (V, C)$, where V is a finite set of *variables*, and C is a finite set of *constraints*. Each constraint $c \in C$ is a pair (s, R) associating a relation $R \in \Gamma$ with a *scope* $s \in V^{\mathrm{ar}(R)}$.

We say an assignment $h : V \to D$ *satisfies* a constraint $c = (s, R)$ if $h(s) \in R$. The problem is then to decide whether there exists an assignment that satisfies all constraints $c \in C$.

Example 2.3. The graph 2-colouring problem can be seen as an example of a CSP. Let $\Gamma = \{R\}$ be a constraint language over the domain $D = \{0, 1\}$, containing a single binary relation $R = \{(0, 1), (1, 0)\}$. Then, CSP(Γ) is equivalent to the graph 2-colouring problem.

A graph $G = (V, E)$, is encoded as a CSP(Γ) instance $I = (V, C)$ where the set of vertices of G and the set of variables of I are the same, and the set of constraints C is constructed from the set of edges E by adding for each edge $(u, v) \in E$ a constraint $((u, v), R)$ to C. It is easy to check that a graph G is 2-colourable if and only if the CSP(Γ) instance I is satisfiable: The domain values 0 and 1 correspond to the two different ways to colour a vertex, and any assignment satisfying a constraint of the form $((u, v), R)$ must colour the vertices u and v with different colours.

Example 2.4. Another example of a CSP is 3-SAT. For any triple $(i, j, k) \in \{0, 1\}^3$, let us define the relation $R_{ijk} = \{0, 1\}^3 \backslash (i, j, k)$. Let Γ then be the constraint language over the domain $D = \{0, 1\}$ that contains the eight relations R_{ijk} for $i, j, k \in \{0, 1\}$. Then, CSP(Γ) is equivalent to 3-SAT.

Observe that the relations R_{ijk} are such that each relation is equivalent to a clause with three literals, with the indices i, j, and k indicating which literals are negated. For instance, the following equivalence holds:

$$(x, y, z) \in R_{010} \Leftrightarrow (x \vee \neg y \vee z).$$

We can now see that any 3-SAT instance can be encoded as an instance $I = (V, C)$ of CSP(Γ). The set of variables V is simply taken as the set of variables in the 3-SAT formula. The set of constraints C then contains for each clause a constraint $((x, y, z), R_{ijk})$, where x, y, and z are the variables occuring in the clause, and i, j, and k are chosen depending on which literals are negated. This construction ensures that the 3-SAT formula is satisfiable if and only if the CSP instance I is as well.

We see from the two examples above that depending on which constraint language we choose, the difficulty of solving CSP(Γ) can be either rather easy (in the 2-colouring case), or NP-hard (in the 3-SAT case). Indeed, as we see later, the complexity of CSP(Γ) is completely determined by algebraic properties of Γ.

Another view on constraint satisfaction problems is to interpret CSPs as *structural homomorphism* problems. A homomorphism between two relational structures $\mathbf{A}$ and $\mathbf{B}$ is a mapping h of elements of $\mathbf{A}$ to elements of $\mathbf{B}$, such that all relations are preserved, i.e. $h : \mathrm{dom}(\mathbf{A}) \to \mathrm{dom}(\mathbf{B})$, and for every m-ary relation symbol R, it holds $R^{\mathbf{A}}(x_1, \ldots, x_m) \Rightarrow R^{\mathbf{B}}(h(x_1), \ldots, h(x_m))$ for all $x_1, \ldots, x_m \in \mathrm{dom}(\mathbf{A})$. Given a constraint language Γ over a domain D, it is not difficult to see that an instance $I = (V, C)$ of CSP(Γ) is essentially an instance of the homomorphism problem. Namely, let $\mathbf{I}$ be the structure with V as its universe, with relations $R^{\mathbf{I}}$, where $s \in R^{\mathbf{I}}$ if and only if (s, R) is a constraint in C. Observe that I is satisfiable if and only if there is a homomorphism from $\mathbf{I}$ to $\mathbf{\Gamma} = (D, \Gamma)$. This follows immediately from the definitions: A satisfying assignment for I is a mapping $h : V \to D$, such that for all constraints $c = (s, R)$ it holds $R(h(s))$. This is exactly a homomorphism from $\mathbf{I}$ to $\mathbf{\Gamma}$. In light of this connection, it is occasionally convenient to parameterise CSPs by a relational structure instead of a constraint language. That is, we use CSP($\mathbf{\Gamma}$) where $\mathbf{\Gamma}$ is a structure with universe D and relations Γ to denote the constraint problem CSP(Γ) over the domain D.

There are several ways one can imagine extending the framework of CSPs to be able to express optimisation problems. For instance, instead of deciding whether all constraints are satisfiable at the same time, one could ask for the maximal number of satisfiable constraints. This leads to the class of so-called MAXCSP problems. One popular example is MAX-3-SAT, which is the maximisation version of 3-SAT. An even finer approach is to further distinguish between different assignments to the same constraint, and give a value to "how much" a constraint is satisfied by a particular assignment. These ideas lead us to the framework of *constraint optimisation*, also called *valued CSPs* (VCSPs) in the literature.

Definition 2.5. Let Γ be a constraint language. An instance of the *valued constraint satisfaction problem* over Γ, or VCSP(Γ), is a pair $I = (V, C)$, where V is a finite set of variables, and C is a finite set of constraints. Each constraint $c \in C$ is a triple (s, f, w) associating a function $f \in \Gamma$ with a scope $s \in V^{\mathrm{ar}(f)}$ and a weight $w \in \mathbb{Z}^+$.

A *solution* to an instance I is an assignment $h : V \to D$. The *cost* of a solution h is defined as

$$\gamma_I(h) := \sum_{(s,f,w)\in C} w \cdot f(h(s)).$$

A solution is *feasible* if its cost is finite. The task is to find a solution that minimises the cost γ_I. For a fixed instance I, we define its *optimal value* as $\mathrm{Val}_I := \min_h \gamma_I(h)$.

A variant of VCSPs that model "pure" optimisation problems are so-called *finite-valued* CSPs. A constraint language Γ is called finite-valued, if every function $f \in \Gamma$ is $\mathbb{Q}$-valued, instead of $\mathbb{Q}_\infty$. In particular this means that any solution to a finite-valued CSP is feasible. It turns out that the complexity of finite-valued CSPs is in some ways easier to analyse, as we see later in Chapter 5. When it is preferable to make the distinction clear, we call the more general case *general-valued* CSPs.

When talking about the complexity of problems, it is often more convenient to also phrase VCSPs as decision problems. Abusing notation, we will use VCSP(Γ) to also denote the set of pairs (I, t) such that I is an instance with $\mathrm{Val}_I \leq t$. The decision problem is then, given any pair (I, t), to decide whether $(I, t) \in \mathrm{VCSP}(\Gamma)$. We call the value for t the *threshold* of the instance I. In line with the usual definition, a *reduction* from VCSP(Γ) to VCSP(Δ) is a mapping $r : (I, t) \mapsto (J, u)$ that maps any instance (I, t) of VCSP(Γ) to an instance (J, u) of VCSP(Δ), such that $(I, t) \in \mathrm{VCSP}(\Gamma)$ if and only if $(J, u) \in \mathrm{VCSP}(\Delta)$. We call a reduction r *optimum preserving* if for all instances (I, t), $r(I, t) = (J, t)$, i.e. r always maps instances to instances that have the same threshold. Otherwise, we say r is a *threshold reduction*.

Note that in the setting of constraint *satisfaction*, we were only concerned with *relational* constraint languages, while for *optimisation* problems, we dropped this restriction. Occasionally we call the more general case *valued* constraint languages, to emphasise the distinction. Observe that for relational Γ, CSP(Γ) is clearly just a special case of VCSP(Γ), where an instance is satisfiable if and only if there is some assignment of finite cost.

Example 2.6. A *cut* in an undirected graph $G = (V, E)$ is a partition of the vertex set V into two parts, and the *value* of a cut is the number of edges connecting vertices of different parts. The MAXCUT problem is to determine, given a graph G, and a value t, whether G admits a cut of value at least t.

This can be formulated as a finite-valued VCSP. Let $\Gamma = \{f\}$ be the valued constraint language over the domain $D = \{0, 1\}$ containing a single function $f : \{0, 1\}^2 \to \{0, 1\}$, defined by

$$f(a, b) = \begin{cases} 0 & \text{if } a \neq b, \\ 1 & \text{otherwise.} \end{cases}$$

Then, VCSP(Γ) encodes the MAXCUT problem.

A MAXCUT instance (G, t) can be encoded as an instance $I = (V, C, t')$ of VCSP($\{f\}$) as follows. The set of variables V is again just the set of vertices of G, and each edge (u, v) in G creates a constraint of the form $((u, v), f, 1)$ in the constraint set C. Note that f is defined such that an edge connecting vertices of different parts costs 0, while an edge connecting vertices belonging to the same part invokes a cost of 1. Hence, the graph G has a cut of value at least t, if and only if I has an assignment of cost at most $|E| - t$. Consequently, we can set the threshold for the VCSP instance as $t' = |E| - t$, completing the construction.

Example 2.7. A *vertex cover* in an undirected graph $G = (V, E)$ is a set of vertices, such that every edge in E is incident to at least one vertex in the set. The VCOVER problem is to determine, given a graph G, and a threshold t, whether G contains a vertex cover of size at most t.

This fits again into our VCSP framework. Consider the constraint language $\Gamma = \{f, g\}$ with $f : \{0, 1\}^2 \to \{0, 1\}$ defined as

$$f(a, b) = \begin{cases} \infty & \text{if } a = b = 0, \\ 0 & \text{otherwise,} \end{cases}$$

and $g : \{0, 1\} \to \{0, 1\}$ defined as $g(a) = a$. Then, VCSP(Γ) encodes the VCOVER problem.

A VCOVER instance is represented as an instance of VCSP(Γ) as follows: Given a graph G and a threshold t, we encode it as an instance $I = (V, C, t')$ with V being the set of vertices. Furthermore, we add the constraint $((u, v), f, 1)$ for each edge (u, v) in G, as well as the constraint $(v, g, 1)$ for every vertex v. The threshold t' is set to the threshold of the vertex cover problem t.

In this construction, an assignment $V \to \{0, 1\}$ is interpreted as a vertex cover containing exactly those vertices that are assigned to 1. The constraints of the form $((u, v), f, 1)$ enforce that the assignment is actually a valid vertex cover, as edges that are not incident to any vertex in the cover are penalised by a cost of infinity. The other constraints $(v, g, 1)$ then ensure that an assignment with minimal cost corresponds to a minimal vertex cover, as each vertex included in the cover adds a value of 1 to the overall cost.

From our examples we see that both CSPs and VCSPs are NP-hard in general. It is also easy to see that both problems lie in NP, as one can simply guess the satisfying or optimal assignment non-deterministically. Still, many constraint problems are in fact much easier to solve than its NP-completeness would suggest, and the search for a classification of these easy cases has been an ongoing quest in the field of complexity theory. One of the main open problems (see Remark 2.9) in this quest is the resolution of the *dichotomy conjecture* formulated by Feder and Vardi [48].

Conjecture 2.8. *For every finite relational constraint language* Γ*, either* $\mathrm{CSP}(\Gamma)$ *is in* P*; or it is* NP*-complete.*

Recall that for general problems, Ladner's theorem [80] states that under the assumption $\mathrm{P} \neq \mathrm{NP}$, there exists problems in NP that are neither in P, nor NP-complete. If the dichotomy conjecture were true, then CSPs would be a large natural class that does not contain any problems of intermediate difficulty. Since the conjecture was posed, several special cases of the dichotomy have been confirmed [92, 61, 21, 20, 11]. However, the status of the general case remains unresolved.

Remark 2.9. Very recently, several claims [23, 87, 102] have appeared that claim to have proved the conjecture. At the time of writing, none of these proofs have been peer-reviewed yet, and we will treat the conjecture as still open.

One of the most succesful approaches towards understanding the complexity of CSPs so far has come from an algebraic perspective. This approach has been essentially initiated by Cohen, Jeavons and Krokhin [25] who observed that the complexity of $\mathrm{CSP}(\Gamma)$ is closely related to certain algebraic closure properties of Γ. Much of the recent progress in the field has come from refining our understanding of this algebraic connection.

2.3.1 Polymorphisms and algebras

We give a short overview over the essential concepts and ideas behind the algebraic approach to the study of constraint problems.

We start with the concepts that relate to constraint satisfaction problems, and see later how they are carried over to the optimisation setting. Much of the content here (and more) can also be found for example in the surveys [28, 8, 79].

A basic concept in studying the complexity of problems are *reductions*. Informally, if one can "simulate" every instance of problem A as an instance of problem B, then problem A *reduces* to problem B, and we can conclude that problem B is at least as hard as problem A. With respect to CSPs, one particular notion of "simulation", so-called *primitive positive definitions* (pp-definitions), plays a central role.

Definition 2.10. A relation $R \subseteq D^m$ is *primitive positive definable* (pp-definable) from a relational constraint language Γ, if for some $k \geq 0$, R can be defined as

$$R(a_1, \ldots, a_m) \equiv \exists x_1 \ldots \exists x_k \phi(a_1, \ldots, a_m, x_1, \ldots, a_k),$$

where ϕ is a finite conjunction of relations of Γ, and equalities.

Example 2.11. Let Γ be the constraint language for 3-SAT from Example 2.4. The logical OR-relation, $R_\vee = \{(a, b) \in \{0, 1\}^2 \mid a \vee b\}$, is pp-definable from Γ, by

$$R_\vee(a, b) \equiv \exists x R_{000}(a, b, x) \wedge R_{001}(a, b, x).$$

We can easily generalise this concept to whole constraint languages, instead of single relations. For two relational constraint languages Γ and Δ, we say Δ is pp-definable from Γ, if every relation in Δ is pp-definable from Γ. The following theorem shows that pp-definability is in fact a sort of "simulation".

Theorem 2.12 ([69])**.** *Let Γ and Δ be two relational constraint languages, such that Δ is pp-definable from Γ. Then,* $\mathrm{CSP}(\Delta)$ *is reducible in polynomial time to* $\mathrm{CSP}(\Gamma)$*.*

The notion of pp-definability gives us a notion of *closure*: Given a relational constraint language Γ, we can define a new constraint language $\langle\Gamma\rangle_{pp}$ as the smallest set that contains all relations that are pp-definable from Γ. Clearly, $\mathrm{CSP}(\Gamma)$ and $\mathrm{CSP}(\langle\Gamma\rangle_{pp})$ have the same complexity (with regards to polynomial-time tractability), as well as every other constraint language whose closure is also $\langle\Gamma\rangle_{pp}$. This motivates us to study the complexity of $\mathrm{CSP}(\Gamma)$ in terms of properties of $\langle\Gamma\rangle_{pp}$. Interestingly, we can also understand $\langle\Gamma\rangle_{pp}$ in algebraic terms. For this, we introduce the notion of *polymorphisms*.

Definition 2.13. Let $R \subseteq D^m$ be any m-ary relation. An operation $f : D^k \to D$ is a *polymorphism* of R, if for any choice of k m-tuples $\{t_1 = (t_{11}, \ldots, t_{1m}), \ldots, t_k = (t_{k1}, \ldots, t_{km})\}$ with $t_1, \ldots, t_k \in R$, applying f component-wise to $t_1, \ldots, t_k$ yields a tuple in R. That is,

$$(f(t_{11}, \ldots, t_{k1}), \ldots, f(t_{1m}, \ldots, t_{km})) \in R.$$

Example 2.14. Let $R := \{(0, 1), (1, 0)\}$ be the inequality relation over $\{0, 1\}$. Moreover, let $\mathrm{maj} : \{0, 1\}^3 \to \{0, 1\}$ be the ternary majority operation over $\{0, 1\}$. That is, $\mathrm{maj}(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$. The operation maj is a polymorphism of R: Choosing any three pairs among $\{(0, 1), (1, 0)\}$, the component-wise application of maj simply yields the pair that is chosen the most often.

For a negative example, recall the relation $R_{000} = \{0, 1\}^3 \backslash (0, 0, 0)$ from Example 2.4. Here, maj is not a polymorphism of R_{000}. Take the three tuples $t_1 = (0, 1, 0)$, $t_2 = (1, 0, 0)$, and $t_3 = (0, 0, 1)$. The component-wise application of majority yields

$$(\mathrm{maj}(0, 1, 0), \mathrm{maj}(1, 0, 0), \mathrm{maj}(0, 0, 1)) = (0, 0, 0),$$

which is not in the relation R_{000}.

We define the set $\mathrm{Pol}(\Gamma)$ as the set of all functions that are polymorphisms of *all* relations in Γ.

$$\mathrm{Pol}(\Gamma) := \{f \mid \forall R \in \Gamma : f \text{ is a polymorphism of } R\}.$$

Conversely, for a given set of operations O, we define the set $\mathrm{Inv}(O)$ as the set of relations for which *all* operations in O are polymorphisms.

$$\mathrm{Inv}(O) := \{R \mid \forall f \in O : f \text{ is a polymorphism of } R\}.$$

Remark 2.15. Note that $\langle\Gamma\rangle_{pp}$ is always an *infinite* constraint language. This leads to some issues regarding the encoding of instances. Namely, for any instance I of $\mathrm{CSP}(\Gamma)$, there is a relation in $\langle\Gamma\rangle_{pp}$ that is the conjunction of all constraints in I. Hence, the instance I could be encoded as an instance of $\mathrm{CSP}(\langle\Gamma\rangle_{pp})$ that contains only a single constraint.

To ensure that tractability does not depend on the way in which the relations are encoded, we only consider the complexity of *finite* constraint languages, often defined as finite subsets of infinite closures. Hence, from here on, constraint languages are assumed to be finite, unless they are defined as infinite sets in the first place.

The following theorem states that the pp-closure of Γ is defined by the set of polymorphisms of Γ.

Theorem 2.16 ([25])**.** *Let Γ be a relational constraint language. Then, $\langle\Gamma\rangle_{pp} = \mathrm{Inv}(\mathrm{Pol}(\Gamma))$.*

As a direct consequence, we have that two constraint languages with the same set of polymorphisms also have the same closure, and thus also the same complexity.

Corollary 2.17. *Let Γ and Δ be two relational constraint languages. If $\mathrm{Pol}(\Gamma) \subseteq \mathrm{Pol}(\Delta)$, then $\mathrm{CSP}(\Delta)$ is reducible in polynomial time to $\mathrm{CSP}(\Gamma)$.*

Proof. If $\mathrm{Pol}(\Gamma) \subseteq \mathrm{Pol}(\Delta)$, then by definition, $\mathrm{Inv}(\mathrm{Pol}(\Delta)) \subseteq \mathrm{Inv}(\mathrm{Pol}(\Gamma))$. By Theorem 2.16, this means that $\langle\Delta\rangle_{pp} \subseteq \langle\Gamma\rangle_{pp}$. Hence, Δ is pp-definable from Γ, and the claim follows from Theorem 2.12. □

Remark 2.18. For finer grained notions of complexity, the polynomial time reduction in Theorem 2.12 (and consequently also in the above corollary) can be improved to a logspace-reduction (see [69]). In the context of descriptive complexity, the reduction can also be formulated in Datalog, as was shown in [5].

As we see, the complexity of $\mathrm{CSP}(\Gamma)$ is determined by the set of polymorphisms $\mathrm{Pol}(\Gamma)$. Hence, in the algebraic approach the main objects of study are these sets $\mathrm{Pol}(\Gamma)$. For any relational constraint language Γ, $\mathrm{Pol}(\Gamma)$ has the following properties:

- $\mathrm{Pol}(\Gamma)$ contains all *projections*. That is, for every $n \in \mathbb{N}$, and $i \leq n$, the function
$$\pi_i^n(x_1, \ldots, x_n) = x_i$$
is in $\mathrm{Pol}(\Gamma)$.

- Pol(Γ) is closed under *composition*. That is, for any n-ary operation $g \in \text{Pol}(\Gamma)$, and n many k-ary operations $f_1, \ldots, f_n \in \text{Pol}(\Gamma)$, the k-ary operation

$$g(f_1, \ldots, f_n)(x_1, \ldots, x_k) = g(f_1(x_1, \ldots, x_k), \ldots, f_n(x_1, \ldots, x_k))$$

  is in Pol(Γ).

Sets of operations that contain all projections, and are closed under compositions are called *clones*. We sometimes refer to Pol(Γ) then as the *clone of polymorphisms* of Γ. If Γ is a constraint language over the domain D, we can associate an *algebra* $\text{Alg}(\Gamma) := (D, \text{Pol}(\Gamma))$ with Γ, where the element set is D, and Pol(Γ) is the set of operations. This formulation has proven quite fruitful, as there are many results connecting algebraic properties of Alg(Γ) to the computational complexity of CSP(Γ). In Section 2.6 we summarise a number of such results that are relevant to our work.

In more recent work [78, 30, 95, 98, 96, 32, 79], much of the algebraic groundwork for constraint satisfaction has been carried over to the setting of optimisation. Similar to the relational case, for valued constraint languages Γ, we can again find operations on Γ that preserve the complexity of VCSP(Γ). The analogue to pp-definability here is the notion of *expressibility*.

Definition 2.19. Let Γ be a valued constraint language over some domain D. A function $f : D^m \to \mathbb{Q}_\infty$ is *expressible* over Γ, if there is some instance $I_f = (V_f, C_f)$ of VCSP(Γ) and a list of variables $v_1, \ldots, v_m \in V_f$, such that

$$f(x_1, \ldots, x_m) = \min_{\{h : V_f \to D \mid h(v_i) = x_i\}} \gamma_{I_f}(h),$$

where γ_{I_f} is the cost function of I_f as defined in Definition 2.5.

Note that the list of variables $v_1, \ldots, v_m$ may contain duplicates, i.e. it may be that $v_i = v_j$ for some distinct i and j. It is then possible that there is no assignment h such that $h(v_i) = x_i$ for all i. We define the minimum over an empty set to be ∞.

Example 2.20. Let $\Gamma = \{f, g\}$ be the constraint language from Example 2.7, and let $p : \{0, 1\} \to \{0, 1\}$ be defined as $p(a) = 1 - a$. The function p is expressible over Γ. To see this, take the instance $I = (V, C)$ of VCSP(Γ) with two variables x and y, and two constraints $((x, y), f, 1)$ and $(y, g, 1)$. We can quickly check that

$$p(a) = \min_{\{h : V \to D \mid h(x) = a\}} \gamma_I(h).$$

The notion of pp-definability can be seen as a special case of expressibility, only for relational constraint languages. It is not difficult to prove that if a relation R is pp-definable from Γ, then it is also expressible in Γ (where relations are viewed as functions).

Lemma 2.21. *Let $R : D^m \to \{0, \infty\}$ be an m-ary relation over D that is pp-definable from some constraint language Γ. Then, R is also expressible in Γ.*

Proof. If R is pp-definable from Γ, then by definition it can be written as

$$R(a_1, \ldots, a_m) \equiv \exists x_1 \ldots \exists x_k \phi(a_1, \ldots, a_m, x_1, \ldots, x_k),$$

where ϕ is a conjunction using relations in Γ and equalities. Given ϕ, we define the following instance $I_\phi = (V, C)$ of VCSP(Γ). The variables of the instance are exactly the unbounded variables of ϕ, which we denote by $V = \{v_1, \ldots, v_{m+k}\}$. The constraints of I are constructed form the terms of the conjunction ϕ. That is, each relational term of the form $Q(s)$, where $Q \in \Gamma$ and $s \subseteq V^{\mathrm{ar}(Q)}$, creates a constraint in C of the form $(s, Q, 1)$. Finally, given I_ϕ, we can write R as

$$R(a_1, \ldots, a_m) = \min_{h \in H} \gamma_{I_\phi}(h),$$

where the set H is defined as

$$H := \{h : V \to D \mid \forall i, j \in [m] : h(v_i) = a_i \wedge (v_i = v_j) \in \phi \Rightarrow h(v_i) = h(v_j)\}.$$

Namely, we consider only those assignments that also respect all equalities in ϕ.

To see that this representation of R is accurate, observe that $\min_{h \in H} \gamma_{I_\phi}(h)$ yields 0 if and only if the conjunction ϕ is satisfiable when when fixing $v_i \mapsto a_i$, and is ∞ otherwise. □

Example 2.22. Let Γ be the constraint language for 3-SAT, and we consider again the logical OR-relation $R_\vee = \{(0,1), (1,0), (1,1)\}$. In Example 2.11 we have seen that $R_\vee$ is pp-definable from Γ. In fact, $R_\vee$ is also expressible over Γ. Take the instance $I = (V, C)$ of VCSP(Γ) with $V = \{x, y, z\}$ and the two constraints $((x, y, z), R_{000}, 1)$ and $((x, y, z), R_{0,0,1}, 1)$. It is easy to check that

$$R_\vee(a, b) = \min_{\{h : V \to D \mid h(x) = a;\ h(y) = b\}} \gamma_I(h).$$

Recall here that $R_\vee$ can be viewed as a function $R_\vee : \{0,1\}^2 \to \{0, \infty\}$.

In addition to expressibility, the operations of *scaling* and *translation* by appropriate constants also preserve the complexity of VCSP(Γ). A function f is obtained from g by scaling and translation if f can be defined as $f = a \cdot g + b$ for some choice of $a, b \in \mathbb{Q}, a > 0$. Again, these operations give us a notion of *closure*: For a valued constraint language Γ, we define $\langle\Gamma\rangle$ as the smallest set of functions that contains Γ and is closed under expressibility, scaling, and translation. In the literature, a set of operations closed under expressibility, scaling and translation is also sometimes called a *weighted relational clone*, and the closure $\langle\Gamma\rangle$ is then called the weighted relational clone of Γ.

Theorem 2.23 ([30])**.** *Let Γ and Δ be two valued constraint languages, such that $\Delta \subseteq \langle\Gamma\rangle$. Then,* VCSP($\Delta$) *is reducible in polynomial time to* VCSP(Γ)*.*

Similar to the relational case, there is an alternative, algebraic view on $\langle\Gamma\rangle$. As in the relational case, the notion of polymorphisms turns out again to be a powerful tool, albeit with some adaptations. In particular two generalisations, namely the so-called *fractional* and *weighted polymorphisms*, have been established as the key concepts in the world of constraint optimisation.

Definition 2.24. Let $f : D^m \to \mathbb{Q}_\infty$ be an m-ary function over D. We associate with it its *feasibility relation*

$$\mathrm{Feas}(f) := \{x \in D^m \mid f(x) \neq \infty\}.$$

We say that an operation $o : D^k \to D$ is a *polymorphism* of f, if it is a polymorphism in the relational sense of $\mathrm{Feas}(f)$. The notion of a *fractional polymorphisms* is defined as follows.

Definition 2.25. Let $f : D^m \to \mathbb{Q}_\infty$, and let $\mathrm{Pol}^{(k)}(f)$ denote the set of all k-ary polymorphisms of f. A k-ary *fractional polymorphism* of f is a function $w : \mathrm{Pol}^{(k)}(f) \to \mathbb{Q}^+$, such that

- $\sum_{o \in \mathrm{Pol}^{(k)}(f)} w(o) = 1$,
- for any tuples $t_1, \ldots, t_k \in \mathrm{Feas}(f)$, it holds

$$\sum_{o \in \mathrm{Pol}^{(k)}(f)} w(o) \cdot f(o(t_1, \ldots, t_k)) \leq \frac{1}{k} \sum_{i \in [k]} f(t_i).$$

Again, o is applied component-wise, i.e. for $t_i = (t_{i1}, \ldots, t_{im})$ we have

$$o(t_1, \ldots, t_k) = (o(t_{11}, \ldots, t_{k1}), \ldots, o(t_{1m}, \ldots, t_{km})).$$

Example 2.26. Let $f : \{0,1\}^2 \to \mathbb{Q}$ be the function defined as

$$f(a,b) = \begin{cases} 0 & \text{if } a = b, \\ 1 & \text{otherwise.} \end{cases}$$

Furthermore, let $\min : \{0,1\}^2 \to \{0,1\}$ and $\max : \{0,1\}^2 \to \{0,1\}$ be the binary minimum and maximum operations respectively.

Observe first that since f is finite-valued, any pair in $\{0,1\}^2$ is in $\mathrm{Feas}(f)$. Consequently, every operation over $\{0,1\}$ is a polymorphism of f, in particular min and max. Now let $w : \mathrm{Pol}^{(2)}(f) \to \mathbb{Q}^+$ be the mapping that maps the operations min and max to $1/2$ each, and is 0 otherwise. We can check that w is a binary fractional polymorphism of f. For example, if we take the two pairs $t_1 = (0,1), t_2 = (0,0)$, on the left side we would have

$$\frac{1}{2} \cdot f(\min(t_1, t_2)) + \frac{1}{2} \cdot f(\max(t_1, t_2)) = \frac{1}{2} f(0,0) + \frac{1}{2} f(0,1) = \frac{1}{2},$$

while the right hand side would also sum up to $1/2$.

Intuitively, a k-ary fractional polymorphism can be seen as a probability distribution over the set of k-ary polymorphisms of a function f. A fractional polymorphism w assigns to each operation in $\mathrm{Pol}^{(k)}(f)$ some weight, with the total sum of the weights being 1. Similarly, a probability distribution over $\mathrm{Pol}^{(k)}(f)$ assigns each operation a probability, and the sum over all probabilities must be 1 as well. From this probabilistic perspective, the left hand side of the inequality in Definition 2.25 can be interpreted as the expected value of $f(o(t_1, \ldots, t_k))$ when drawing the operation o with probability $w(o)$. The inequality then reads

$$\mathbb{E}_{o \sim w}[f(o(t_1, \ldots, t_k))] \leq \mathrm{avg}\{f(t_1), \ldots, f(t_k)\}.$$

A closely related notion to fractional polymorphisms are *weighted polymorphisms*. In fact, both concepts are equivalent, and while fractional polymorphisms are more easily motivated through their similarity to probability distributions, weighted polymorphisms are more convenient to work with in certain cases.

Definition 2.27. Let C be a clone of operations, and let $C^{(k)}$ denote the set of all k-ary operations in C. A k-ary *weighting* of C is a mapping $w : C^{(k)} \to \mathbb{Q}$, such that

- $\sum_{o \in C^{(k)}} w(o) = 0$,
- $w(o) < 0$ if and only if o is a projection.

The *support* of w is the set of operations to which w assigns positive weight,

$$\mathrm{supp}(w) := \{o \in C^{(k)} \mid \ w(o) > 0\}.$$

Definition 2.28. Let $f : D^m \to \mathbb{Q}_\infty$, and let $C \subseteq \mathrm{Pol}(f)$. A k-ary weighting $w : C^{(k)} \to \mathbb{Q}$ is called a k-ary *weighted polymorphism* of f over C if for any k m-tuples $t_1, \ldots, t_k \in \mathrm{Feas}(f)$, we have

$$\sum_{o \in C^{(k)}} w(o) \cdot f(o(t_1, \ldots, t_k)) \leq 0.$$

Again, o is applied component-wise.

As mentioned, weighted polymorphisms and fractional polymorphisms describe equivalent concepts, and can be transformed into each other. The transformation between the two kinds is essentially a rescaling of weights, which can be seen when considering the inequality in Definition 2.25. Observe that $f(t_i)$ can be rewritten as $f(\pi_i^k(t_1, \ldots, t_k))$, where π_i^k is simply the projection of the i-th component out of k. Moving the right hand side to the left, we obtain

$$\sum_{o \in \mathrm{Pol}^{(k)}(f)} w(o) \cdot f(o(t_1, \ldots, t_k)) - \sum_{i \in [k]} \frac{1}{k} \cdot f(\pi_i^k(t_1, \ldots, t_k)) \leq 0.$$

Since the projections π_i^k are contained in $\mathrm{Pol}^{(k)}(f)$, we can merge the two sums, which results in an inequality similar to the one in Definition 2.28.

Lemma 2.29 ([79]). *Given a function $f : D^m \to \mathbb{Q}_\infty$, and a k-ary fractional polymorphism w of f, there is a k-ary weighted polymorphism w' of f with the same (non-projection) supports, and vice versa.*

Similar to the relational setting, we can associate with a constraint language Γ, the set of weighted polymorphisms of all functions in Γ, $\mathrm{wPol}(\Gamma)$.

$$\mathrm{wPol}(\Gamma) := \{w \mid \forall f \in \Gamma : w \text{ is a weighted polymorphism of } f \text{ over } \mathrm{Pol}(f)\}.$$

In the converse, for a set of weightings W over a fixed clone C, we denote by $\mathrm{wInv}(W)$ the set of all functions that have all weightings in W as weighted polymorphisms.

$$\mathrm{wInv}(W) := \{f \mid \forall w \in W : w \text{ is a weighted polymorphism of } f \text{ over } C\}.$$

Then, an analogue of Theorem 2.16 is obtained from [30].

Theorem 2.30 ([30]). *For any finite constraint language Γ, $\mathrm{wInv}(\mathrm{wPol}(\Gamma)) = \langle\Gamma\rangle$.*

We see that, in a very similar way to the case for constraint satisfaction, the complexity of constraint optimisation problems $\mathrm{VCSP}(\Gamma)$ is closely tied to the set of (weighted) polymorphisms of Γ. A particularly useful notion that connects the two worlds of satisfaction and optimisation is the *positive clone* of a constraint language Γ over domain D, defined as

$$\mathrm{Pol}^+(\Gamma) := \Pi_D \cup \bigcup_{w \in \mathrm{wPol}(\Gamma)} \mathrm{supp}(w),$$

where Π_D denotes the set of all projections over the domain D.

Note that $\mathrm{Pol}^+(\Gamma)$ is a set of (non-weighted) polymorphisms, and is a subset of $\mathrm{Pol}(\Gamma)$. In fact, $\mathrm{Pol}^+(\Gamma)$ turns out to also be a clone (hence the name), i.e. it contains all projections and is closed under composition.

Lemma 2.31 ([78]). *For any constraint language Γ, $\mathrm{Pol}^+(\Gamma)$ is a clone.*

Since $\mathrm{Pol}^+(\Gamma)$ is a clone of polymorphisms, $\mathrm{Inv}(\mathrm{Pol}^+(\Gamma))$ is an infinite relational constraint language. It turns out that the complexity of the optimisation problem $\mathrm{VCSP}(\Gamma)$ is closely connected to the complexity of the satisfaction problem $\mathrm{CSP}(\Delta)$ when $\Delta \subset \mathrm{Inv}(\mathrm{Pol}^+(\Gamma))$. We will review some of the results that establish this connection later in Section 2.6.

2.4 Linear and semidefinite optimisation

We give a brief introduction to linear and semidefinite programming. A more complete exposition on this topic can be found in textbooks, such as [59, 18].

Linear or semidefinite programming refer to classes of optimisation problems that fall into the greater framework of *convex optimisation*. In such problems, one

is generally interested in optimising a given cost function $c : \mathbb{Q}^V \to \mathbb{Q}$ over some convex set $\mathcal{F}$, called the *feasible region*, embedded in the vector space $\mathbb{Q}^V$. In the case of linear programming, $\mathcal{F}$ is defined by a finite set of linear constraints, i.e. $\mathcal{F}$ is the intersection of a finite set of linear half-spaces, or a polyhedron. On the other hand, in semidefinite programs, the feasible region consists of an intersection of a polyhedron with the cone of positive semidefinite matrices. We will work with the following definitions.

Definition 2.32. Let V and M be non-empty sets. A *linear program* (LP) is given by a cost vector $c \in \mathbb{Q}^V$, a matrix $A \in \mathbb{Q}^{M \times V}$, and a vector $b \in \mathbb{Q}^M$. Its feasible region is defined as

$$\mathcal{F}_{A,b} := \{x \in \mathbb{Q}^V \mid \langle A_i, x \rangle \leq b_i \text{ for all } i \in M\}.$$

While the search space of linear programs are vectors $x \in \mathbb{Q}^V$, in semidefinite programming, we optimise over the set of positive semidefinite $V \times V$ matrices. Hence, our vector space here is $\mathbb{Q}^{V \times V}$.

Definition 2.33. Let V and M be non-empty sets. A *semidefinite program* (SDP) is given by a cost matrix $C \in \mathbb{Q}^{V \times V}$, a $\mathbb{Q}^{V \times V}$-valued vector $\mathcal{A} \in \mathbb{Q}^{M \times (V \times V)}$, and a vector $b \in \mathbb{Q}^M$. Its feasible region is defined as

$$\mathcal{F}_{\mathcal{A},b} := \{X \in \mathbb{Q}^{V \times V} \mid X \succeq 0, \langle \mathcal{A}_i, X \rangle \leq b_i, \text{ for all } i \in M\},$$

where $X \succeq 0$ denotes that X is a positive semidefinite matrix.

We sometimes call sets that can be defined as feasible regions of an SDP a *positive semidefinite set*. The above definition covers SDPs that are in the so-called *conic standard form*. Sometimes it is more convenient to also consider SDPs in their *inequality standard form*. In this form, an SDP is instead given by a matrix $Z \in \mathbb{Q}^{M \times M}$, a matrix-valued vector $\mathcal{Y} \in \mathbb{Q}^{V \times (M \times M)}$, and an objective vector $c \in \mathbb{Q}^V$. The feasible region is then defined as

$$\mathcal{F}_{\mathcal{Y},Z} := \{x \in \mathbb{Q}^V \mid Z + \sum_{v \in V} x_v \mathcal{Y}_v \succeq 0\}.$$

The two standard forms can be easily converted into each other by adding, substituting, and rearranging variables. Hence, depending on the situation, we will use whichever representation is most convenient.

Classically, in the context of convex optimisation, there are two main problems of interest, namely the *optimisation* problem and the *separation* problem. Informally, in the optimisation problem, we are to find a point in the feasible region that maximises (or minimises) a given linear objective function c, or to determine that no such point exists. The separation problem is to determine whether a given point x is feasible, and if not, to produce a witness in form of a hyperplane that separates x from the feasible region.

As a technical point, the optimal solution to a semidefinite program may not be rational, even when all coefficients are. Since we can express solutions only up to a finite precision, we allow an additive error to be specified in the input. This additional error term distinguishes between the strong and weak formulations of the two main problems in optimisation.

Definition 2.34. Let $\mathcal{F} \subseteq \mathbb{Q}^V$ be a convex set, and let $\delta \in \mathbb{Q}$ with $\delta > 0$. A vector $y \in \mathbb{Q}^V$ is *δ-close* to $\mathcal{F}$, if $d(y, \mathcal{F}) \leq \delta$. It is called *$\delta$-maximal* with regards to an objective vector $c \in \mathbb{Q}^V$, if $\langle c, y\rangle + \delta \geq \max_{x\in\mathcal{F}}\langle c, x\rangle$.

Definition 2.35. Let V be a non-empty set. Given a vector $c \in \mathbb{Q}^V$ and a convex set $\mathcal{F} \subseteq \mathbb{Q}^V$, the *strong optimisation problem* is to either find an element $y = \operatorname{argmax}_{x\in\mathcal{F}}\langle c, x\rangle$, or to determine that $\mathcal{F}$ is empty, or that $\max_{x\in\mathcal{F}}\langle c, x\rangle$ is unbounded.

In the *weak optimisation problem* we are given an additional error parameter $\delta > 0$. The goal is to find an element y such that y is δ-close to $\mathcal{F}$ and δ-maximal with regards to c, or to determine that $\max_{x\in\mathcal{F}}\langle c, x\rangle$ is unbounded.

Definition 2.36. Let V be a non-empty set. Given a vector $y \in \mathbb{Q}^V$ and a convex set $\mathcal{F} \subseteq \mathbb{Q}^V$, the *strong separation problem* is the problem of determining either that $y \in \mathcal{F}$, or finding a vector $s \in \mathbb{Q}^V$ with $\langle s, y\rangle > \max\{\langle s, x\rangle \mid x \in \mathcal{F}\}$ and $\|s\|_\infty = 1$.

In the *weak separation problem*, we are given an additional parameter $\delta > 0$, and are looking to determine that either y is δ-close to $\mathcal{F}$, or to find a vector $s \in \mathbb{Q}^V$, such that $\langle s, y\rangle + \delta > \max\{\langle s, x\rangle \mid x \in \mathcal{F}\}$ and $\|s\|_\infty = 1$.

The relationship between the optimisation and separation problem of a given convex set is well-studied and is most prominently expressed by Grötschel, Lovász, and Shrijver [58] as being polynomial time equivalent. More precisely, with the additional assumptions that the set $\mathcal{F}$ is *full-dimensional* ($\mathcal{F}$ has positive volume in $\mathbb{Q}^V$) and *bounded* ($\mathcal{F}$ is contained within a ball of finite radius), the weak optimisation problem for $\mathcal{F}$ is solvable in polynomial time if, and only if, the corresponding weak separation problem is solvable in polynomial time.

Theorem 2.37 ([59])**.** *Let V be a non-empty set, and $\mathcal{F} \subseteq \mathbb{Q}^V$ a full-dimensional convex set that is located inside the ball $\mathcal{B}(0, R)$ for some known value R. The weak optimisation problem on $\mathcal{F}$ is solvable in polynomial time if its weak separation problem is solvable in polynomial time.*

A classic result in complexity theory is due to Khachiyan [70], showing that in the special case where $\mathcal{F}$ is the feasible region of a linear program, the strong versions of both optimisation and separation can be solved in polynomial time, without the additional assumptions above. The main algorithm for solving LPs in polynomial time, as well as the main ingredient of the polynomial time reduction in the above theorem is the *ellipsoid method* (see [59]).

In the broadest terms, the setup for the ellipsoid method is as follows. The goal is to find a feasible point inside some convex set $\mathcal{F}$, which is only given through

a so-called *separation oracle*. That is, instead of asking for the explicit constraints defining $\mathcal{F}$, we have access to a black box solver for the separation problem for $\mathcal{F}$. The ellipsoid method is then an algorithm that repeatedly calls the separation oracle to shrink the search space of finding a point in $\mathcal{F}$. If the assumptions of Theorem 2.37 hold, then this process is guaranteed to terminate within a number of steps polynomial in the size of the representation of $\mathcal{F}$ and R. Hence, if the separation problem for $\mathcal{F}$ is solvable in polynomial time, so is locating a feasible point in $\mathcal{F}$. Finally, finding an optimum can be reduced to finding any feasible solution, by adding the cost vector as a constraint for $\mathcal{F}$.

In the case of semidefinite programs, a simple (strong) separation oracle is given by Algorithm 1. Note that this algorithm assumes that we can compute eigenvalues and eigenvectors with infinite precision. Realistically we will have to work with finite precision approximations that can be obtained in polynomial time. In Chapter 5, we will formulate a slightly modified algorithm that serves as a weak separation oracle that can be expressed in fixed-point logic with counting (FPC).

Recently, Anderson et al. [3, 4] have shown that the ellipsoid method can be expressed in FPC, in the case of polyhedra. In particular, the authors establish that if a separation oracle for polyhedra can be defined in FPC, then there is also an FPC-formula that solves the corresponding optimisation problem. In Chapter 5, we will extend their construction to positive semidefinite sets. Together with the FPC-definable weak separation oracle, this yields a definability result for semidefinite programs.

Algorithm 1 Separation oracle for semidefinite programs

Input: $\mathcal{A} = \{A_1, \ldots, A_m \in \mathbb{Q}^{V \times V}\}$, $b \in \mathbb{Q}^m$, $Y \in \mathbb{Q}^{V \times V}$.
Output: Solves separation problem on $\mathcal{F}_{\mathcal{A},b}$ and Y.

```
1: function SEPARATION(A,b,Y):
2:     if there is A_i ∈ A such that ⟨A_i, Y⟩ > b_i then
3:         return 1/‖A_i‖_∞ · A_i
4:     Compute Eigenvalues {λ_1, ..., λ_|V|} of Y
5:     if there is λ_i < 0 then
6:         v ← Eigenvector corresponding to λ_i
7:         return (−1)/‖vv^T‖_∞ · vv^T
8:     return ACCEPT
```

2.4.1 Constraint problems as linear programs

While classical linear programs are useful to express many optimisation problems, for combinatorial problems, the frameworks of *integer linear programs* (or even *0–1 linear programs*) are more suitable. The problem formulation is the same as given in Definition 2.32, only the domain is restricted to integers (or just $\{0, 1\}$), instead of the rational numbers.

Most relevant to us, general constraint satisfaction problems can be represented in this framework. Given an instance $I = (V, C)$ of VCSP(Γ), it is equivalent to the following 0–1 program.

$$\min \sum_{c \in C} \sum_{x \in D^{\mathrm{ar}(s)}} \lambda_{c,x} \cdot w \cdot f(x) \quad \text{where } c = (s, f, w), \text{ s.t.}$$

$$\sum_{x \in D^{\mathrm{ar}(s)}; x_i = a} \lambda_{c,x} = \mu_{s_i,a} \qquad \forall c \in C, a \in D, i \in [\mathrm{ar}(s)] \tag{2.1}$$

$$\sum_{a \in D} \mu_{v,a} = 1 \qquad \forall v \in V \tag{2.2}$$

$$\lambda_{c,x} = 0 \qquad \forall c \in C, x \notin \mathrm{Feas}(f) \tag{2.3}$$

$$\tag{2.4}$$

The program contains variables $\lambda_{c,x}$ for every $c \in C$ with $c = (s, f, w)$ and $x \in D^{\mathrm{ar}(s)}$, and $\mu_{v,a}$ for every $v \in V$ and $a \in D$. A solution that sets a variable $\lambda_{c,x}$ to 1 then corresponds to an assignment that assigns the scope of the constraint c to the tuple x. In order to maintain consistency of the assignment between constraints, the variable $\mu_{v,a}$ encodes whether the variable v is assigned the value a. The constraints $\sum_{a \in D} \mu_{v,a} = 1$ enforce that each variable v is assigned to exactly one value $a \in D$, and the constraints $\sum_{x \in D^{\mathrm{ar}(s)}} \lambda_{c,x} = \mu_{s_i,a}$ make sure that that if the scope s of constraint c is assigned to the tuple x, then each component s_i is assigned to the value $a = x_i$. Note that this also implies that for any fixed constraint c there is exactly one variable $\lambda_{c,x}$ that is assigned 1. The objective is then to minimise the cost of the assignment.

In constrast to rational linear programs, solving 0–1 LPs is NP-hard, and formulating a CSP as a 0–1 LP instance does not make it easier to solve exactly. Still, there are some ways one can exploit the similarity between integer and rational LPs. In particular, if we are interested in *approximate* solutions rather than exact ones, we can relax the *integrality constraints* of the 0–1 LP, namely the implicit constraints $\lambda, \mu \in \{0, 1\}$, to also allow rational assignments to the variables. The resulting LP instance is called the *basic linear program relaxation* of I, or BLP(I). Since this is a rational LP now, it can be solved in polynomial time. In general, the optimal value of BLP(I) gives only an underestimate of the optimal value of I. However, for certain languages Γ, the optimum of BLP(I) is exactly the optimal solution to I, yielding an exact algorithm to solve VCSP(Γ) for those constraint languages. For the case that Γ is finite-valued, Thapper and Živný [96] established a complete characterization of those cases, which is discussed later in Section 2.6.

The idea of relaxing certain constraints of an integer LP to obtain a rational LP (or SDP) is extended further through the notion of relaxation hierarchies.

2.4.2 Relaxation hierarchies

As illustrated in the previous section, one of the applications of linear or semidefinite programming is to find approximation algorithms for hard combinatorial problems. For a large class of problems, namely those that can be expressed as integer programs, a generic way to find approximations is to drop the integrality condition so that a rational solution can be efficiently computed. The value of the optimal rational solution then serves as an approximation to the optimal value of the integer problem. The concept of relaxation hierarchies extends this idea further. Instead of solving the basic relaxation, these hierarchies define a sequence of linear or semidefinite programs that provide increasingly finer approximations to the original integer solution. This is achieved by adding, at each level of the hierarchy, additional constraints to the basic relaxation that are preserved under the original integer program, but may cut away rational solutions not in the convex hull of integer solutions.

In this thesis, we primarily consider the *Lasserre hierarchy* [82], which for a given 0–1 linear program defines a sequence of semidefinite programs.

Definition 2.38. Let V, U be sets and $\mathcal{K} := \{x \in \mathbb{Q}^V \mid Ax \geq b\}$ a polytope given by $A \in \mathbb{Q}^{U \times V}, b \in \mathbb{Q}^U$.

For a vector $y \in \mathbb{Q}^{\wp(V)}$, and an integer t with $1 \leq t \leq |V|$, we define the *t-th moment matrix* of y, $M_t(y)$ as the $\wp_t(V) \times \wp_t(V)$-matrix with entries

$$M_t(y)_{I,J} := y_{I \cup J}, \text{ for } |I|, |J| \leq t.$$

Similarly, the *t-th moment matrix of slacks* of y, A, b, and some $u \in U$ is given by

$$S_t^u(y)_{I,J} := \sum_{v \in V} A_{u,v} y_{I \cup J \cup \{v\}} - b_u y_{I \cup J}, \text{ for } |I|, |J| \leq t.$$

Finally, the *t-th level of the Lasserre hierarchy* of $\mathcal{K}$, $\mathrm{Las}_t(\mathcal{K})$ is the set defined by

$$\begin{aligned} \mathrm{Las}_t(\mathcal{K}) \quad := \quad & \{y \in \mathbb{Q}^{\wp_{2t+1}(V)} \mid y_\emptyset = 1, M_t(y) \succeq 0, \\ & S_t^u(y) \succeq 0 \text{ for all } u \in U\}. \end{aligned}$$

We write $\mathrm{Las}_t^\pi(\mathcal{K}) := \{y_{\{v\}}, v \in V \mid y \in \mathrm{Las}_t(\mathcal{K})\}$ for the projection of $\mathrm{Las}_t(\mathcal{K})$ onto the original variables.

The general usage of the Lasserre hierarchy is as follows. Assume we have a 0–1 program where the feasible region is defined as $\mathcal{K} \cap \{0,1\}^V$. Now, instead of optimizing over the integer region, we can define $\mathrm{Las}_t(\mathcal{K})$ for some level t, and solve the corresponding SDP. For a fixed constant t, this SDP has a polynomial number of new variables, and the (weak) optimum can be obtained in polynomial time (with repsect to the bit size of $\mathcal{K}$ and the error parameter δ). This optimum, when projected down onto the original variables, serves as an approximation to the optimum in $\mathcal{K} \cap \{0,1\}^V$.

The following basic properties of the Lasserre hierarchy establish that $\mathrm{Las}_t(\mathcal{K})$ is indeed a relaxation of $\mathcal{K} \cap \{0,1\}^V$. We write $\mathcal{K}^*$ for the polytope that is defined by the convex hull of the integer points in $\mathcal{K}$, i.e. $\mathcal{K}^* := \mathrm{conv}(\mathcal{K} \cap \{0,1\}^V)$.

Lemma 2.39 ([90, 29])**.** *Let $\mathcal{K} = \{x \in \mathbb{Q}^V \mid Ax \geq b\}$. Then,*

1. *$\mathcal{K}^* \subseteq \mathrm{Las}_t^\pi(\mathcal{K})$ for all $t \in \{1, \ldots, |V|\}$.*
2. *$\mathrm{Las}_0^\pi(\mathcal{K}) \supseteq \mathrm{Las}_1^\pi(\mathcal{K}) \supseteq \ldots \supseteq \mathrm{Las}_{|V|}^\pi(\mathcal{K})$.*
3. *$\mathrm{Las}_0^\pi(\mathcal{K}) \subseteq \mathcal{K}$, and $\mathcal{K}^* = \mathrm{Las}_{|V|}^\pi(\mathcal{K})$.*

Definition 2.40. Let $I = (c, \mathcal{K})$ be a 0–1 linear program with an objective vector $c \in \mathbb{Q}^V$ optimizing over a feasible region $\mathcal{K} \cap \{0,1\}^V$. We say that I is *captured* at the t-th level of the Lasserre hierarchy if the projection of the t-th level Lasserre relaxation coincides with the convex hull of integer solutions in the direction of c, i.e. $\max_{x \in \mathcal{K}^*} \langle c, x \rangle = \max_{x \in \mathrm{Las}_t^\pi(\mathcal{K})} \langle c, x \rangle$. We write $l(I)$ for the minimum t, such that I is captured at the t-th level.

For a class of 0–1 linear programs $\mathcal{C}$, we say that $\mathcal{C}$ is captured at the t-th level, if every program in $\mathcal{C}$ is captured at the t-th level of the Lasserre hierarchy.

We denote by $L_{\mathcal{C}}(n)$ the function that maps an integer n to the first level t at which every 0–1 program in $\mathcal{C}$ with size n is captured. That is, $L_{\mathcal{C}}(n) := \max_{I \in \mathcal{C}; |V| \leq n} l(I)$.

We see that at a sufficiently high level, namely at most at level $t = |V|$, any 0–1 program is captured by the t-th level Lasserre relaxation. In those cases, the optimum of the Lasserre set yields not only an approximate optimum, but is the exact optimal value of the original 0–1 problem.

In Chapter 5, we establish a dichotomy for finite-valued VCSPs with respect to $L_{\mathcal{C}}(n)$: For every finite-valued Γ, $\mathcal{C} = \mathrm{VCSP}(\Gamma)$, either $L_{\mathcal{C}}(n) = 0$, meaning $\mathrm{VCSP}(\Gamma)$ is solved by the basic linear program relaxation; or $L_{\mathcal{C}}(n) \in \Omega(n)$.

2.5 Logic

In this section we provide a brief summary of the basic concepts from mathematical logic that we will need in this thesis. The content here can also be found in many standard textbooks, such as [44, 83].

In the field of logic, mathematical objects such as graphs, matrices, or CSP instances, are modelled as *relational structures* over some *vocabulary* (also called *signature*).

Definition 2.41. A *vocabulary* or *signature* τ is a finite sequence of relation and constants symbols $(R_1, \ldots, R_k, c_1, \ldots, c_l)$, where each relation symbol R_i has a fixed arity $\mathrm{ar}(R_i) \in \mathbb{N}$.

A *structure* $\mathbf{A}$ over the vocabulary τ (also called a *τ-structure*) is the tuple $\mathbf{A} = (\mathrm{dom}(\mathbf{A}), R_1^{\mathbf{A}}, \ldots, R_k^{\mathbf{A}}, c_1^{\mathbf{A}}, \ldots, c_l^{\mathbf{A}})$, consisting of a non-empty set $\mathrm{dom}(\mathbf{A})$, called the

universe of $\mathbf{A}$, and relations $R_i^{\mathbf{A}} \subseteq \mathrm{dom}(\mathbf{A})^{\mathrm{ar}(R_i)}$ and constants $c_j^{\mathbf{A}}$ for all $i \in [k]$ and $j \in [l]$.

Members of $\mathrm{dom}(\mathbf{A})$ are called *elements* of $\mathbf{A}$, and the *size* of $\mathbf{A}$ is the cardinality of its universe.

Example 2.42. The vocabulary of graphs consists of a single binary relation symbol, i.e. $\tau_{\mathrm{Graph}} = \{E\}$ with $\mathrm{ar}(E) = 2$. A particular graph $\mathbf{G}$ is simply a structure over this vocabulary, $\mathbf{G} = (V, E^{\mathbf{G}})$, with some fixed universe V and edge relation $E^{\mathbf{G}}$.

We often also consider structures with elements from multiple distinct domains. These so-called *multi-sorted* structures are defined as follows.

Definition 2.43. A *k-sorted vocabulary* or *signature* is a vocabulary

$$\tau = (R_1, \ldots, R_n, c_1, \ldots, c_l)$$

where every relation or constant symbol S has a *type*, denoted $\mathrm{type}(S)$, such that

- for a relation symbol R of arity m, $\mathrm{type}(R) \in [k]^m$;
- and if c is a constant symbol, $\mathrm{type}(c) \in [k]$.

A *structure* over a k-sorted vocabulary τ is is the tuple

$$\mathbf{A} = (\mathrm{dom}(\mathbf{A}), R_1^{\mathbf{A}}, \ldots, R_n^{\mathbf{A}}, c_1^{\mathbf{A}}, \ldots, c_l^{\mathbf{A}}),$$

where

- the universe $\mathrm{dom}(\mathbf{A}) = U_1 \dot\cup \ldots \dot\cup U_k$ consists of the disjoint union of k sets;
- $R^{\mathbf{A}} \subseteq U_{t_1} \times \ldots \times U_{t_m}$ for each relation symbol R of arity m with $\mathrm{type}(R) = (t_1, \ldots, t_m)$.
- $c^{\mathbf{A}} \in U_t$ for each constant symbol c with $\mathrm{type}(c) = t$.

Unless specified otherwise, we assume structures to have a *finite* universe and write $\mathrm{fin}[\tau]$ for the set of all finite τ-structures.

Given a signature τ, a logic L defines a collection of *formulas* $\mathrm{L}[\tau]$, as well as a *satisfaction relation* $\models_{\mathrm{L}}$. The formulas of a logic are usually defined recursively. For example, consider *first-order logic* (FO). The atomic formulas of $\mathrm{FO}[\tau]$ are formulas of the form $s = t$, where s and t are element variables or constant symbols in τ; and $R(t_1, \ldots, t_m)$, where $t_1, \ldots, t_m$ are element variables or constant symbols in τ, and R is a relation symbol in τ with arity m. The set of all formulas $\mathrm{FO}[\tau]$ is then defined by closing the set of atomic formulas under the operations of negation, conjunction, disjunction, and first-order universal and existential quantification. Formulas with no free variables are called *sentences*. Throughout this thesis, we consider several

different extensions and restrictions of FO, however, quantifiers remain exclusively first-order.

For a given τ-structure $\mathbf{A}$ and a formula ϕ of $\mathrm{L}[\tau]$, an *assignment* f of ϕ in $\mathbf{A}$ is a mapping of the free variables of ϕ to elements of $\mathbf{A}$. The satisfaction relation relates structures with formulas and assignments: We write $\mathbf{A} \models_{\mathrm{L}} \phi[f]$ to say that $\mathbf{A}$ satisfies the formula ϕ under the assignment f. For the sake of convenience, if $x_1, \dots, x_m$ are the free variables of ϕ, we write $\phi[a_1, \dots, a_m]$ to denote the sentence $\phi[f]$ with the assignment $f : x_i \mapsto a_i$. If ϕ does not have free variables, we can instead write $\mathbf{A} \models_{\mathrm{L}} \phi$. We often omit the subscript if the logic L is clear from context. Naturally, the definition of this relation depends on the semantics of the logic L. As the logics we consider in this thesis are all derived from first-order logic in some fashion, we will refrain from formally describing complete semantics. Instead, we introduce logics as extensions or restrictions of first-order logic, and point out the relevant differences.

Example 2.44. Let $\tau_{\mathrm{Graph}} = (E)$ be the vocabulary of graphs. The following sentence in $\mathrm{FO}[\tau_{\mathrm{Graph}}]$ is satisfied by a graph $\mathbf{G}$ if and only if $\mathbf{G}$ contains a (directed) triangle.

$$\exists x_1, x_2, x_3 \quad (x_1 \neq x_2) \wedge (x_2 \neq x_3) \wedge (x_1 \neq x_3) \wedge E(x_1, x_2) \wedge E(x_2, x_3) \wedge E(x_3, x_1).$$

One of the main themes of this thesis is to use logic as a means to measure the complexity of problems, in particular CSPs. To expand on this concept further, we define what it means for a class of structures to be *definable* in a certain logic L. For a formula ϕ in $\mathrm{L}[\tau]$, we define the class of finite *models* of ϕ as

$$\mathrm{Mod}(\phi) := \{\mathbf{A} \in \mathrm{fin}[\tau] \mid \mathbf{A} \models \phi\}.$$

Definition 2.45. Let τ be a signature. A class of structures $\mathcal{C} \subseteq \mathrm{fin}[\tau]$ is said to be *definable* in a logic L, if there is a sentence $\phi \in \mathrm{L}[\tau]$, such that $\mathcal{C} = \mathrm{Mod}(\phi)$.

The notion of definability gives us a notion of expressive power of logics: If every class that is definable in a logic L_1 is also definable in another logic L_2, then we can say that L_2 has at least the expressive power of L_1. In this case, we write $\mathrm{L}_1 \leq \mathrm{L}_2$. Likewise, we use $\mathrm{L}_1 \lneq \mathrm{L}_2$ if $\mathrm{L}_1 \leq \mathrm{L}_2$ and there is some class definable in L_2 but not in L_1.

Often, extending an existing logic with more syntactical elements, such as high-order quantifiers or additional operators, results in a strictly more expressive logic, but not always. The consideration of syntactical richness as a complexity resource is the key idea in the field of *descriptive complexity*. Descriptive complexity explores the complexity of classes of problems based on their definability, and aims to establish connections between computational resources needed to solve certain problems and the syntactical richness of logics necessary to define them.

2.5.1 Interpretations

A common tool when comparing the complexity of different classes of problems are reductions. To this end, it is necessary to consider mappings between structures of different signatures. In logic, this is achieved by *interpretations*.

Definition 2.46. Let τ and σ be two signatures, and L a logic. An m-ary L-*interpretation* of τ in σ is a sequence of formulas in $\mathrm{L}[\sigma]$ consisting of

- a formula $\delta(x)$;
- a formula $\varepsilon(x, y)$;
- for each relation symbol $R \in \tau$ of arity k, a formula $\phi_R(x_1, \ldots, x_k)$;
- for each constant symbol $c \in \tau$, a formula $\gamma_c(x)$,

where each x, y, or x_i is an m-tuple of free variables. We call m the *width* of the interpretation.

We say that an interpretation Θ associates a τ-structure $\mathbf{B}$ to a σ-structure $\mathbf{A}$ if there is a surjective map h from the m-tuples $\{a \in \mathrm{dom}(\mathbf{A})^m \mid \mathbf{A} \models \delta[a]\}$ to $\mathbf{B}$ such that:

- $h(a_1) = h(a_2)$ if, and only if, $\mathbf{A} \models \varepsilon[a_1, a_2]$;
- $R^{\mathbf{B}}(h(a_1), \ldots, h(a_k))$ if, and only if, $\mathbf{A} \models \phi_R[a_1, \ldots, a_k]$;
- $h(a) = c^{\mathbf{B}}$ if, and only if, $\mathbf{A} \models \gamma_c[a]$.

Note that an interpretation Θ associates a τ-structure to $\mathbf{A}$ only if ε defines an equivalence relation on $\mathrm{dom}(\mathbf{A})^m$ that is a congruence with respect to the relations defined by the formulas ϕ_R and γ_c. In such cases, however, $\mathbf{B}$ is uniquely defined up to isomorphism and we write $\Theta(\mathbf{A}) := \mathbf{B}$.

Throughout this thesis, all interpretations Θ associate a τ-structure to every $\mathbf{A}$, and we take care that ε is always a congruence. Moreover, ε is frequently simply defined as the usual equality on a_1 and a_2. In these instances, we omit the explicit definition of ε.

We are now ready to introduce the notion of L-*reductions*.

Definition 2.47. Let C_1 and C_2 be two classes of σ- and τ-structures respectively. We say that C_1 L-*reduces* to C_2 if there is an L-interpretation Θ of τ in σ, such that $\Theta(\mathbf{A}) \in C_2$ if and only if $\mathbf{A} \in C_1$, and we write $C_1 \leq_{\mathrm{L}} C_2$.

It is not difficult to see that L-formulas compose with L-interpretations in the following sense. Given an interpretation Θ of τ in σ and a τ-formula ϕ, we can define a σ-formula ϕ' such that $\mathbf{A} \models \phi'$ if, and only if, $\Theta(\mathbf{A}) \models \phi$. This is achieved simply by replacing the relation and constant symbols in ϕ by the corresponding σ-formulas

in the interpretation Θ (with suitable renaming of variables to avoid capture). Note that if ϕ uses k variables, the composition ϕ' may contain up to $m \cdot k$ many variables, where m is the width of Θ. Likewise, interpretations themselves compose. That is, given interpretations Θ of τ in σ, and Σ of σ in ρ, we can obtain an interpretation Θ' of τ in ρ by composition: Θ' consists of the functions of Θ where the relation symbols of σ are instead replaced by the corresponding ρ-formulas in Σ.

2.5.2 Representation

In order to discuss definability of constraint problems, we need to define how instances of these problems are represented as relational structures. Here, we fix a number of signatures that define the relational representation of most common objects.

Numbers. We represent an integer z as a relational structure in the following way. First, decompose $z = s \cdot x$ into its sign $s \in \{-1, 1\}$ and its absolute value $x \in \mathbb{N}$, and let $b \geq \lceil \log_2(x) \rceil$. We represent z as the structure $\mathbf{z}$ with universe $\mathrm{dom}(\mathbf{z}) = [b]$ over the vocabulary $\tau_{\mathbb{Z}} = \{X, S, <\}$, where

- $X^{\mathbf{z}}$ is a unary relation that encodes the bit representation of x, i.e. $X^{\mathbf{z}} = \{k \in [b] \mid \mathrm{BIT}(x, k) = 1\}$;
- $S^{\mathbf{z}}$ is a unary relation where $S^{\mathbf{z}} = \emptyset$ indicates that $s = 1$, and $s = -1$ otherwise;
- and $<$ is interpreted as the usual linear order on $[b]$.

In a similar vein, we represent a rational number $q = s \cdot \frac{x}{d}$ by a structure $\mathbf{q}$ over the domain $\tau_{\mathbb{Q}} = \{X, D, S, <\}$, where the additional relation $D^{\mathbf{q}}$ encodes the binary representation of the denominator d in the same way as above.

Vectors and matrices. In order to represent vectors and matrices over integers or rationals, we use multi-sorted universes. Let T be a non-empty set, and let v be a vector of integers indexed by T. We represent v as a structure $\mathbf{v}$ with a two-sorted universe with an index sort T, and bit sort $[b]$, where $b \geq \lceil \log_2(m) \rceil$, $m = \max_{t \in T} |v_t|$, over the vocabulary $(X, S, <)$. Here,

- S has arity 2, and $S^{\mathbf{v}}(t, \cdot)$ encodes the sign of the integer v_t for $t \in T$ just like in the integer case;
- X is a binary relation interpreted as $X^{\mathbf{v}} = \{(t, k) \in T \times [b] \mid \mathrm{BIT}(v_t, k) = 1\}$;
- and $<$ is again interpreted as the usual linear order on $[b]$.

Similarly, in order to represent matrices $M \in \mathbb{Z}^{T_1 \times T_2}$, we have three-sorted universes with two sorts of index sets T_1 and T_2, or simply a single index set that consists of pairs. The matrix M is then represented as a structure $\mathbf{M}$ in the same way as a vector, only S and X are now ternary relation symbols, instead of binary. Tensors of

even higher dimensions, such as matrix-valued vectors, are represented in a similar fashion, simply by increasing the number of index sets, as well as the arity of the relation X. The generalization to rationals carries over from the numbers case. We write τ_{vec} to denote the vocabulary for vectors over $\mathbb{Q}$, τ_{mat} for the vocabulary for matrices over $\mathbb{Q}$, and τ_{tens} for the vocabulary for matrix-valued vectors.

Linear and semidefinite Programs. We represent linear or semidefinite programs in their respective standard forms in the following way. An instance of a linear program in standard form is given by a constraint matrix $A \in \mathbb{Q}^{M\times V}$, and vectors $b \in \mathbb{Q}^M, c \in \mathbb{Q}^V$. Hence, we represent it as a structure with a three-sorted universe with the two index sets V and M, and a bit sort $[b]$ over the vocabulary

$$\tau_{\text{LP}} = \tau_{\text{mat}} \dot\cup \tau_{\text{vec}} \dot\cup \tau_{\text{vec}} = (X_A, D_A, S_A, X_b, D_b, S_b, X_c, D_c, S_c, <),$$

where

- X_A,D_A, and S_A are ternary symbols, encoding the entries of the matrix A;
- X_b,D_b, and S_b are binary symbols, encoding the entries of the vector b;
- X_c,D_c, and S_c are binary symbols, encoding the entries of the vector c;
- and $<$ is again interpreted as the usual linear order on $[b]$.

Likewise, a semidefinite program in conic standard form is specified by a matrix-valued vector $\mathcal{A} \in \mathbb{Q}^{M\times(V\times V)}$, an objective matrix $C \in \mathbb{Q}^{V\times V}$, and a vector $b \in \mathbb{Q}^M$. This is represented as a structure over $\tau_{\text{SDP}} = \tau_{\text{tens}} \dot\cup \tau_{\text{mat}} \dot\cup \tau_{\text{vec}}$. Here, the universe is again three-sorted, with the two index sets V and M plus a bit sort. Note that $X_{\mathcal{A}}$ is a quaternary relation, and X_C and X_b are ternary and binary relations.

Sometimes it is more convenient to consider an SDP in inequality standard form, which is specified by a matrix $Z \in \mathbb{Q}^{M\times M}$, a matrix-valued vector $\mathcal{Y} \in \mathbb{Q}^{V\times(M\times M)}$ and an objective vector $c \in \mathbb{Q}^V$. Note that the vocabulary for both representations are the same. Furthermore, as the conversion between the two standard forms essentially consists of rearranging vector entries into suitable block diagonal matrices and vice versa, the conversion can be realised in FPC. We avoid describing the corresponding FPC-interpretation in detail, as it would be quite lengthy without much instructive value.

Constraint problems. For a fixed domain D, and a relational constraint language Γ, we can represent an instance of CSP(Γ) in a natural way. Namely, the vocabulary $\tau_{\text{CSP}(\Gamma)}$ consists of the symbols of all relations in Γ. An instance $I = (V, C)$ is then represented as the $\tau_{\text{CSP}(\Gamma)}$-structure $\mathbf{I} = (V, (R^{\mathbf{I}})_{R\in\Gamma})$, where the universe is the set of variables V, and $s \in R^{\mathbf{I}}$ if there is a constraint $c = (s, R)$ in the constraint set C.

For the more general valued CSPs, we define the vocabulary as $\tau_{\text{VCSP}(\Gamma)} = \{(R_f)_{f\in\Gamma}, W, <\}$. An instance $I = (V, C)$ is then represented as a structure $\mathbf{I}$ with

a three-sorted universe: A sort for variables V; a sort of constraints C; and a bit sort $[b]$ for some sufficiently large b. The relation $R_f^{\mathbf{I}} \subseteq V^{\mathrm{ar}(f)} \times C$ then contains a tuple (s, c) if C contains a constraint of the form (s, f, w). Similarly, the relation $W^{\mathbf{I}} \subseteq C \times [b]$ encodes the weight of each constraint $c = (s, f, w)$ in the relational representation of (non-negative) integers, i.e. $W^I(c, \cdot) = \{k \in [b] \mid \mathrm{BIT}(w, k) = 1\}$. Finally, $<$ is again just interpreted as the usual natural order on $[b]$.

From here on, we use the symbols τ_{vec}, τ_{mat}, τ_{LP}, τ_{SDP}, and τ_Γ to refer to the vocabulary of vectors, matrices, linear programs, semidefinite programs, and instances of VCSP(Γ), respectively.

2.5.3 Overview of relevant logics

There are a number of different logics that we encounter throughout this thesis. In the following, we give a short introduction to each of the relevant logics.

Existential positive and finite variable logic. We frequently consider the *existential positive* fragment of first-order logic. This simply refers to the logic obtained by removing the negation operator and universal quantification from FO. We denote this logic by $\exists^+$FO. We generalise this notation also to extensions of FO, i.e. $\exists^+$L refers to the existential positive fragment of the logic L.

Other useful fragments are *finite variable logics*. These are logics obtained by restricting a logic to formulas that use only a fixed number of variables. For instance, for any $k \in \mathbb{N}$, we denote by FO^k the fragment of first-order logic where formulas are allowed to contain at most k variables.

Fixed-point logics. Many of our definability results relate to *fixed-point logic with counting* (FPC). Here, we provide a short definition of this logic, as well as its extension with rank operators, *fixed-point logic with rank* (FPR). For more details we refer to [44, 83, 65].

To define FPC, we first introduce the extension of first-order logic with an *inflationary fixed-point operator*. Let $\phi(R, x)$ be a formula in τ with a k-ary relation variable R, and k element variables $x = (x_1, \ldots, x_k)$. That is, for any *fixed* k-ary relation S, $\phi(S)$ is simply a formula in FO[τ] with free variables $x_1, \ldots, x_k$. Given a τ-structure $\mathbf{A}$, the formula $\phi(R, x)$ defines the following operator

$$F_{\phi,\mathbf{A}} : \wp(\mathrm{dom}(\mathbf{A})^k) \to \wp(\mathrm{dom}(\mathbf{A})^k),$$

mapping a relation $S \subseteq \mathrm{dom}(\mathbf{A})^k$ to

$$F_{\phi,\mathbf{A}}(S) := \{a \in \mathrm{dom}(\mathbf{A})^k \mid \mathbf{A} \models \phi(S)[a]\}.$$

Consider now the following increasing sequence of relations $(X)_i$

$$\begin{aligned} X_0 &:= \emptyset \\ X_{i+1} &:= X_i \cup F_{\phi,\mathbf{A}}(X_i). \end{aligned}$$

The above sequence of relations is increasing in the sense that $X_{i+1} \supseteq X_i$ for all i. We call the limit of this sequence the *inflationary fixed point* of $F_{\phi,\mathbf{A}}$, and denote it by $\mathrm{ifp}(F_{\phi,\mathbf{A}})$. It is evident that if $\mathbf{A}$ is of size n, the limit is reached after at most n^k steps, since there are only n^k many k-tuples over $\mathrm{dom}(\mathbf{A})$.

We use this concept to extend FO with a way to express the inflationary fixed point of formulas. This leads to the definition of *inflationary fixed-point logic* (IFP). This logic is obtained by extending FO with the following formula building rule. If $\phi(R, x)$ is a first-order formula with a k-ary relation variable R, and k element variables $x = (x_1, \ldots, x_k)$, then

$$\psi := [\mathbf{ifp}_{R,x}\phi](t)$$

is also a formula, where $t = (t_1, \ldots, t_k)$ is a k-tuple of terms. The free variables of ψ are the free variables occurring in t and the free variables in ϕ other than those in x. For a τ-structure $\mathbf{A}$, and an assignment f, we define the semantics of this formula by

$$\mathbf{A} \models [\mathbf{ifp}_{R,x}\phi](t)[f] \Leftrightarrow f(t) \in \mathrm{ifp}(F_{\phi,\mathbf{A}}).$$

In the context of constraint problems, we often speak of the logic Datalog. In those cases we simply refer to the existential positive fragment of IFP.

Fixed-point logic with counting is a further extension of IFP with the ability to express the cardinality of definable sets. The logic is a multi-sorted logic with two sorts of variables: *element variables*, which are the same element variables as used in FO, and *number variables*, which range over some initial segment of the natural numbers. We usually write element variables with lower-case Latin letters $x, y, \ldots$ and use lower-case Greek letters $\mu, \eta, \ldots$ to denote number variables.

The atomic formulas of FPC[τ] consist of formulas of the form

- $\mu = \eta$ or $\mu \leq \eta$, where μ, η are number variables;
- $s = t$ where s, t are element variables or constant symbols from τ;
- and $R(t_1, \ldots, t_m)$, where each t_i is either an element variable or a constant symbol from τ, and R is a relation symbol form τ of arity m.

The set of formulas in FPC[τ] is built up from the atomic formulas by the following rules.

- Formulas can be formed under the standard first-order operations of negation, conjunction, disjunction, universal and existential quantification;
- they can be formed using an inflationary fixed-point operator $[\mathbf{ifp}_{R,x}\phi](t)$;
- we can form *counting terms* $\#_x\phi$, where ϕ is a formula and x a variable;
- and we can form formulas of the kind $s = t$ and $s \leq t$ where s, t are number variables or counting terms

Collectively, we refer to element variables and constant symbols as *element terms*, and to number variables and counting terms as *number terms*.

For the semantics, number terms take values in $\{0, \dots, n\}$, where $n = \mathrm{dom}(\mathbf{A})$ and element terms take values in $\mathrm{dom}(\mathbf{A})$. The semantics of atomic formulas, fixed-points and first-order operations are defined as usual, with comparison of number terms $\mu \leq \eta$ interpreted by comparing the corresponding integers in $\{0, \dots, n\}$. Finally, consider a counting term of the form $\#_x \phi$, where ϕ is a formula and x an element variable. Here the intended semantics is that $\#_x \phi$ denotes the number (i.e. the element of $\{0, \dots, n\}$) of elements that satisfy the formula ϕ.

Fixed-point logics play an important role in descriptive complexity theory. A classical result is the Immerman-Vardi theorem, which establishes that fixed-point logic can express all polynomial-time properties of finite ordered structures.

Theorem 2.48 ([44])**.** *Over ordered structures, any polynomial-time computable property is definable in the logic* IFP*.*

As IFP is a fragment of FPC, it follows that also in FPC we can express any polynomial-time decidable relation on any ordered domain. Moreover, it is known that many linear algebra operations can be defined in FPC even on vectors and matrices indexed by unordered sets, such as forming the sum/product of matrices, and even defining the characteristic polynomial of a matrix [36, 65]. In a similar vein, Anderson et al. in [3] have shown that an optimal solution of an explicitly given linear program (as a τ_{LP}-structure) can also be defined in FPC. One of the contributions of this thesis is to generalise the latter result to semidefinite programs.

Theorem 2.49 ([3])**.** *There is an* FPC*-interpretation* Φ *of* $\tau_{\mathbb{Q}} \mathbin{\dot\cup} \tau_{vec}$ *in* τ_{LP} *that does the following:*

Let an instance of a linear program be given by (A, b, c) *with* $A \in \mathbb{Q}^{M \times V}$*,* $b \in \mathbb{Q}^M$*, and* $c \in \mathbb{Q}^V$*. Its feasible region is denoted by* $\mathcal{F}_{A,b}$*. Let* $\mathbf{I}$ *be the relational representation of this LP.*

Then, $\Phi(\mathbf{I})$ *defines a relational representation of* (f, v)*, with* $f \in \mathbb{Q}$*,* $v \in \mathbb{Q}^V$*, such that*

- $f = 1$ *if, and only if,* $\max_{x \in \mathcal{F}_{A,b}} c^T x$ *is unbounded;*
- *If* $\mathcal{F}_{A,b} \neq \emptyset$ *then* $v \in \mathcal{F}_{A,b}$*;*
- *and* $f = 0, v = \mathrm{argmax}_{x \in \mathcal{F}_{A,b}} c^T x$ *otherwise.*

Even though FPC is quite expressive, it has known limitations. Most prominently, FPC can not express solvability of linear equations over finite fields [5]. This has motivated the introduction of fixed-point logic with rank (FPR) [38]. In the same way FPC extends IFP with a counting operator, FPR_p extends it with a *rank operator* $\mathbf{rk}_p$ which is able to express the rank of a given matrix over the prime field GF_p. To understand how the rank operator works, we first explain how matrices over the prime field GF_p are encoded by formulas.

Definition 2.50. Let p be a prime, and let $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_l)$ be tuples of element variables.

- Given a τ-structure $\mathbf{A}$, and a formula ϕ with free variables among x and y, we write $\mathrm{mat}_{x,y}(\phi, \mathbf{A})_p$ to denote the $\mathrm{dom}(\mathbf{A})^k \times \mathrm{dom}(\mathbf{A})^l$ 0–1 matrix over GF_p defined for all $a \in \mathrm{dom}(\mathbf{A})^k$ and $b \in \mathrm{dom}(\mathbf{A})^l$ by

$$\mathrm{mat}_{x,y}(\phi, \mathbf{A})_p(a, b) = 1 \Leftrightarrow \mathbf{A} \models \phi[a, b].$$

- Let $\Phi = (\phi_1, \ldots, \phi_{p-1})$ be a tuple of formulas where the free variables of each formula are among x and y. Given a τ-structure $\mathbf{A}$, we write $\mathrm{mat}_{x,y}(\Phi, \mathbf{A})_p$ to denote the $\mathrm{dom}(\mathbf{A})^k \times \mathrm{dom}(\mathbf{A})^l$ matrix over GF_p defined by

$$\mathrm{mat}_{x,y}(\Phi, \mathbf{A})_p = \sum_{i \in [p-1]} i \cdot \mathrm{mat}_{x,y}(\phi_i, \mathbf{A})_p \qquad \mod p.$$

In FPR_p we then allow the formation of *rank terms*. Namely, if x and y are tuples of element variables, and Φ is a $(p-1)$-tuple of FPR_p formulas, then $\mathbf{rk}_p(x, y)\Phi$ is a formula in FPR_p. The intended semantics are that $\mathbf{rk}_p(x, y)\Phi$ interpreted on a structure $\mathbf{A}$ defines the rank of the matrix $\mathrm{mat}_{x,y}(\Phi, \mathbf{A})_p$ over GF_p. The logic FPR is obtained by extending IFP with all operators $\mathbf{rk}_p$ for any prime p.

Remark 2.51. In [53], Grädel and Pakusa showed that this definition of FPR, where we have for each prime p a fixed operator $\mathbf{rk}_p$, is in fact not sufficiently expressive to capture P. However, for a modified version of FPR that contains a rank operator $\mathbf{rk}_p$ where p is a *variable*, the question remains open. Still, throughout this thesis we work with the former definition of FPR.

With the power of rank operators, FPR can both simulate counting, as well as define solvability of systems of linear equations over any finite field [65]. Hence, FPR is strictly more expressive than FPC. Still, as computing the rank of a matrix is possible in polynomial time, deciding whether a given structure satisfies a given FPR formula remains in polynomial time.

Infinitary logics. Another class of logics we consider are *infinitary logics*. A more detailed introduction to these logics can be found in [86].

For any $k \in \mathbb{N}$, the infinitary logic $\mathcal{L}^k$ is obtained by extending k-variable first-order logic FO^k with conjunction and disjunction operations over arbitrary, potentially infinite sets of formulas. More precisely, if Φ is a set of $\mathcal{L}^k$ formulas, then so are $\bigwedge \Phi$ and $\bigvee \Phi$ $\mathcal{L}^k$ formulas. Here, $\bigwedge \Phi$ (or $\bigvee \Phi$) is satisfied, if the conjunction (or disjunction) of all formulas in Φ are satisfied. We denote by $\mathcal{L}^\omega := \bigcup_{k \in \mathbb{N}} \mathcal{L}^k$ the infinitary logic in which each formula uses a finite number of variables.

The expressive power of $\mathcal{L}^k$ is often limited when it comes to expressing cardinalities in one form or another. For instance, the class of structures with at least k elements is not definable in the logic $\mathcal{L}^{k-1}$. Hence, one commonly considers the

extension of $\mathcal{L}^k$ with *counting quantifiers*, yielding the logic $\mathcal{C}^k$. For every $i \in \mathbb{N}$, we define the counting quantifier $\exists^{\geq i}$ as follows: Given a formula ϕ, and some $i \in \mathbb{N}$, the formula $\exists^{\geq i} x\phi$ is satisfied if and only if there are at least i distinct values for x such that $\phi(x)$ is satisfied. Again, we define $\mathcal{C}^\omega := \bigcup_{k\in\mathbb{N}} \mathcal{C}^k$.

The expressive power of infinitary logics can be related to the power of fixed-point logics. It has been observed in [73, 86] that $\mathcal{L}^\omega$ is in fact strictly more expressive than IFP, while $\mathcal{C}^\omega$ is strictly more expressive than FPC.

Theorem 2.52 ([73, 86])**.** IFP $\leq \mathcal{L}^\omega$ *and* FPC $\leq \mathcal{C}^\omega$.

In some instances it is useful to link the expressive power of infinitary logics to *equivalences* in their respective finitary logics. We write L^k and C^k for the fragments of $\mathcal{L}^k$ and $\mathcal{C}^k$ without infinitary conjunction and disjunction, i.e. $\mathrm{L}^k := \mathrm{FO}^k$ and $\mathrm{C}^k := \mathrm{FOC}^k$, where FOC is simply FO with counting quantifiers. Two finite structures $\mathbf{A}$ and $\mathbf{B}$ are equivalent with respect to a logic L, written as $\mathbf{A} \equiv^{\mathrm{L}} B$, if for all sentences ϕ of L, $\mathbf{A} \models \phi$ if and only if $\mathbf{B} \models \phi$. The following theorem shows that the infinitary logics $\mathcal{L}^k$ and $\mathcal{C}^k$ can be characterised using the equivalences $\equiv^{\mathrm{L}^k}$ and $\equiv^{\mathrm{C}^k}$.

Theorem 2.53 ([44])**.** *Let $\mathscr{C}$ be a class of finite structures.*

- *$\mathscr{C}$ is definable by a sentence in $\mathcal{L}^k$ if and only if $\mathscr{C}$ is closed under $\equiv^{\mathrm{L}^k}$.*
- *$\mathscr{C}$ is definable by a sentence in $\mathcal{C}^k$ if and only if $\mathscr{C}$ is closed under $\equiv^{\mathrm{C}^k}$.*

2.6 Complexity of constraint problems

In this section we give some more context to the main contributions of this thesis, and also summarise a number of known results that we use in later chapters. We cover some classical as well as some more recent results on the complexity of constraint problems. For the sake of conciseness, our focus will be on results that are either used directly in our proofs, or to which we make additional contributions. Hence, we might omit some important results in the field that are however not directly connected to this thesis. For a more comprehensive coverage of the current state of these topics, we refer to the surveys [28, 76, 8, 79].

Despite the amount of attention and study it received since its formulation, the fundamental dichotomy conjecture of Feder and Vardi still remains an open problem to this day (see Remark 2.9). However, while the complexity of constraint problems in the most general case is not completely understood, two large classes of tractable problems have been identified. These two classes are commonly known as constraint problems of *bounded width* [81, 10], and those problems that can be solved by the *few subpowers* algorithm [33, 24, 66, 15]. Notably, every constraint problem currently known to be tractable falls into one of these categories.

As the contributions of this thesis are more related to the bounded width case, we will give a brief exposition on this subclass of problems in this section. Additionally, we summarise some key results from the study of constraint problems.

2.6.1 Cores and constants

For constraint problems (and homomorphisms in general), the *core* of a structure plays an important role. Let Γ be a constraint language over some domain D. Intuitively, if there is some element $a \in D$, such that every instance of VCSP(Γ) can be solved optimally without ever assigning any variable to a, then a can be cut out from the domain without changing the set of optimal solutions. Hence, we say a constraint language Γ over D is a *core*, if for every $a \in D$ there is an instance I_a of VCSP(Γ), such that a appears in every optimal assignment. In technical terms, we require for a core Γ that every unary operation in $\mathrm{Pol}^+(\Gamma)$ is surjective. Lemma 2.56 connects this technical definition to the intuition.

Definition 2.54. For a subset $S \subseteq D$, we denote by $\Gamma[S]$ the constraint language over S obtained by restricting every function in Γ to S. A constraint language Δ is a *sub-language* of Γ, if $\Delta = \Gamma[S]$ for some $S \subseteq D$.

Definition 2.55. Let Γ be a constraint language over D. We say Γ is a *core* if every unary operation in $\mathrm{Pol}^+(\Gamma)$ is surjective. We say Δ is a core of Γ, if Δ is a sub-language of Γ and a core.

Lemma 2.56 ([78])**.** *For a constraint language Γ over D, let $f : D \to D$ be a unary operation in $\mathrm{Pol}^+(\Gamma)$, and let I be an instance of VCSP(Γ). If h is an optimal solution to I, then so is $f \circ h$.*

To better understand how the technical definition of a core relates to intuition, consider what it means for a unary operation $f \in \mathrm{Pol}^+(\Gamma)$ to be surjective. A unary operation $f : D \to D$ is essentially a mapping between domain values. Given a solution $h : V \to D$ to some instance $I = (V, C)$, the composition $f \circ h$ then represents the solution obtained by first assigning the variables to domain values according to h, and then applying the mapping f on those domain values. Lemma 2.56 states that whenever f is an element of $\mathrm{Pol}^+(\Gamma)$, and h is an optimal solution, the solution obtained this way is also optimal. The condition that any unary operation in $\mathrm{Pol}^+(\Gamma)$ must be surjective then means that there is no unary $f \in \mathrm{Pol}^+(\Gamma)$ that maps onto a proper subset of D, while still preserving the optimal value. This in turn implies that there is no optimal solution that uses only a proper subset of D.

The next lemma formalises the idea that VCSP(Γ) is essentially solved by solving the problem over a core of Γ.

Lemma 2.57 ([78])**.** *Let Δ be a core of Γ. For every instance I of VCSP(Γ), let I' be the instance of VCSP(Δ) obtained by replacing every function in Γ by its restriction in Δ. Then, $\mathrm{Val}_I = \mathrm{Val}_{I'}$.*

The following lemma describes a useful property of cores. Intuitively, it means that for a core constraint language Γ, we can find a function f in its closure $\langle\Gamma\rangle$ that in some way recognises the polymorphisms of Γ.

Lemma 2.58 ([78]). *Let Γ be a core constraint language over some domain D. Furthermore, for any $m \in \mathbb{N}$, we define $O_D^{(m)} := D^{|D^m|}$ as the set of $|D^m|$-tuples over D. An element $o \in O_D^{(m)}$ can be seen as an m-ary operation over D, $o : D^m \to D$.*

Then, for every m, there exists a function $f \in \langle\Gamma\rangle$, $f : O_D^{(m)} \to \mathbb{Q}_\infty$, and a rational number q, such that for every $o \in O_D^{(m)}$ the following conditions are satisfied:

(i) $f(o) \geq q$,

(ii) $f(o) < \infty$ *if and only if* $o \in \mathrm{Pol}(\Gamma)$,

(iii) and $f(o) = q$ *if and only if* $o \in \mathrm{Pol}^+(\Gamma)$.

Considering the core of a language Γ is a convenient and useful way to impose more structure on Γ. Another restriction to Γ that is often used in addition to going to the core is to consider the language obtained from adding all constants to Γ. This results in the notion of the *rigid core*. We define it formally in terms of its polymorphisms.

Definition 2.59. A constraint language Γ is called a *rigid core*, if every operation $f \in \mathrm{Pol}(\Gamma)$ is idempotent, i.e. it satisfies the identity

$$f(x, x, \ldots, x) = x.$$

Let Γ be a core language over domain D. Then we denote by Γ_c the constraint language obtained from Γ by adding all unary singleton relations $R_a := \{a\}$ for all $a \in D$. It is easy to see that Γ_c is a rigid core: Any polymorphism f of R_a must satsify $f(a, a, \ldots, a) = a$. Finally, the next lemma allows us to just consider Γ_c when studying the complexity of Γ.

Lemma 2.60 ([78]). $\mathrm{VCSP}(\Gamma_\mathrm{c})$ *is reducible in polynomial time to* $\mathrm{VCSP}(\Gamma)$*, and vice versa.*

In fact the above reduction also carries over when considering the definability of $\mathrm{VCSP}(\Gamma)$. We establish in Chapter 4 that the reduction from $\mathrm{VCSP}(\Gamma_\mathrm{c})$ to $\mathrm{VCSP}(\Gamma)$ can be realised in the logic FPC.

2.6.2 Bounded width

Despite their hardness in the general case, a remarkably large class of constraint satisfaction problems can be solved using a simple polynomial time algorithm. Say, we are given an instance $I = (V, C)$ of $\mathrm{CSP}(\Gamma)$. Observe that for a fixed subset $S \subseteq V$ of size k, there is only a fixed number of assignments $h : S \to D$, namely just $|D|^k$. Furthermore, checking whether a given assignment h violates any constraint of C can be done in time linear to the number of constraints in C. This means, that we can construct in time polynomial in the size of C for every tuple $s \in S^k$ the relation

$$H_{s,C} := \{x \in D^k \mid \exists h : x = h(s) \text{ and } h \text{ satisfies all constraints in C}\},$$

where $h : S \to D$, and $h(s)$ denotes the tuple $(h(s_1), \ldots, h(s_k))$. In words, $H_{s,C}$ contains all tuples to which s can be assigned without violating any constraint of C.

The algorithm for a fixed parameter k then works as follows. Let $C^{(0)}$ be set to the initial set of constraints C. In iteration i, the algorithm constructs for all k-sized subsets $S \subseteq V$, and every k-tuple $s \in S^k$ all the relations $H_{s,C^{(i)}}$. Then, the updated set of constraints $C^{(i+1)}$ is defined by adding all constraints of the form $(s, H_{s,C^{(i)}})$ to $C^{(i)}$. The algorithm stops if at some point some relation $H_{s,C^{(i)}}$ is empty, in which case it returns "unsatisfiable"; or nothing changed within one iteration ($C^{(i+1)} = C^{(i)}$), and it returns "possibly satisfiable".

It is easy to check that for fixed k, the algorithm terminates after at most $|V|^{O(k)}$ steps. In the literature, this algorithm (and similar variants) is commonly referred to as the *local consistency* algorithm (with parameter k). Note that whenever the algorithm returns "unsatisfiable", the instance was indeed unsatisfiable, since adding constraints of the form $(s, H_{s,C})$ clearly does not cut away satisfying assignments. On the other hand, when it returns "possibly satisfiable", it is just that – in general, surviving the local consistency check is only a necessary, but not sufficient condition for satisfiability. The following example illustrates this.

Example 2.61. We consider certain systems of equations over the two-element field, expressed as CSPs. Let $\Gamma = \{R_0, R_1\}$ be over the domain $\{0, 1\}$ with

$$R_0 = \{(x, y, z) \in \{0, 1\}^3 \mid x \oplus y \oplus z = 0\},$$
$$R_1 = \{(x, y, z) \in \{0, 1\}^3 \mid x \oplus y \oplus z = 1\},$$

where $\oplus$ denotes addition modulo 2. The following system is an instance of CSP(Γ).

$$a \oplus b \oplus c = 0$$
$$b \oplus c \oplus d = 0$$
$$d \oplus e \oplus f = 0$$
$$a \oplus e \oplus f = 1.$$

Here, the set of variables is $V := \{a, b, c, d, e, f\}$, and we have four constraints $C = \{((a, b, c), R_0), ((b, c, d), R_0), ((d, e, f), R_0), ((a, e, f), R_1)\}$.

This system is clearly unsatisfiable: The total sum of the right hand sides is equal to 1, while the total sum of the left hand sides is 0, since every variable occurs exactly twice. Still, we can check that running the local consistency algorithm for $k = 3$ terminates in a single iteration, yielding "possibly satisfiable".

In the algorithm, the parameter k determines how broad the scopes of the partial assignments are. It is not difficult to see that for $k = |V|$, the algorithm considers every possible *full* assignment to the variables, and hence only returns "possibly satisfiable" if the instance is actually satisfiable. Still, for some Γ, this situation occurs even for constant values of k, and local consistency becomes a necessary as well as sufficient condition for satisfiability. We call those constraint language Γ to be of *bounded width*.

Definition 2.62. A relational constraint language Γ has *width* k, if CSP(Γ) is solved by the local consistency algorithm with parameter k. We say Γ has *bounded width*, if its width is constant.

The class of bounded width CSPs represents one of the main "islands of tractability" in constraint satisfaction, and finding defining characteristics of this class has been a topic of much research throughout the years. Remarkably, progress in this area has come from many different angles, creating a fruitful intersection between algebra, logic, and complexity.

A connection to logic was established early by Feder and Vardi [48], showing that all bounded-width CSPs can be expressed as Datalog programs, which in turn are equivalent to formulas in the positive existential fragment of fixed-point logic. Since then, many results were born of the algebraic approach to the analysis of CSPs. In particular, classifying algebraic properties of Γ in terms of Hobby and McKenzie's tame congruence theory of finite algebras [64] has proven useful. In this context, Larose and Zádori [81] showed that if the variety generated by Alg(Γ) admits the unary or affine type, then CSP(Γ) is not definable in Datalog, and conjectured the converse. Atserias, Bulatov, and Dawar [5] strengthened this result, showing that those undefinable CSPs even remain undefinable in the stronger logic $\mathcal{C}^\omega$. Later, Barto and Kozik [10] achieved a full algebraic characterisation of the bounded-width case, affirming the previously conjectured converse of the Larose-Zádori result. In their follow-up work, the algebraic condition for bounded width was then shown to be equivalent to a condition now known as the *bounded width condition* (BWC). We summarise the different characterisations of bounded width from the perspectives of algebra, logic, and algorithms in Theorem 2.65.

Definition 2.63. A k-ary ($k \geq 2$) idempotent operation $f : D^k \to D$ is called a *weak near-unanimity* (WNU) operation, if for all $x, y \in D$, it satisfies the identities

$$f(x, y, y, \ldots, y) = f(y, x, y, y, \ldots, y) = \ldots = f(y, y, \ldots, y, x).$$

Definition 2.64. A clone of operations satisfies the *bounded width condition* (BWC) if it contains WNU operations of all but finitely many arities.

Theorem 2.65. *Let Γ be a relational constraint language that is a rigid core. The following are equivalent.*

(i) CSP(Γ) *has bounded width.*

(ii) CSP(Γ) *has width* 3.

(iii) Pol(Γ) *satisfies the BWC.*

(iv) The complement of CSP(Γ) *is definable in* Datalog.

Proof. The case $(i) \Leftrightarrow (ii)$ was shown in [9], $(i) \Leftrightarrow (iii)$ is the result of [81, 10], $(i) \Rightarrow (iv)$ is due to [48] and $(iv) \Rightarrow (iii)$ is due to [5, 81, 10]. □

The connection between bounded width CSPs and the logic Datalog has been known quite early, and the result of Kolaitis and Vardi in [75] pinned down that the complement of a CSP with width k is definable in Datalog using at most k distinct variables. Later, Atserias, Bulatov, and Dawar in [5] showed that there is in fact a gap in terms of definability. There the authors show that those CSP(Γ) whose complements are not definable in Datalog are in fact not definable even in the stronger logic $\mathcal{C}^\omega$. Together with Theorem 2.65 this implies a *dichotomy* on the definability of CSPs.

Theorem 2.66. *Let* Γ *be a relational constraint language. Then,*

- *either the complement of* CSP(Γ) *is definable in* Datalog*;*
- *or* CSP(Γ) *is not definable in* $\mathcal{C}^\omega$*.*

Note that this dichotomy result is consistent with the Feder-Vardi dichotomy conjecture (Conjecture 2.8), but does not solve it: There are CSPs that are not definable in $\mathcal{C}^\omega$, but are still solvable in polynomial time, such as solving systems of linear equations. Nonetheless, this result makes an important step in connecting the descriptive complexity view on CSPs with the results that came from the algebraic approach. We will consider the above result as a starting point, and many of our results presented in this thesis can be seen as extensions from it in one way or another. In particular, we show that a similar dichotomy holds as well if we include constraint optimisation, and also show that this dichotomy has algorithmic implications as well.

The notion of bounded width in the satisfaction setting is closely tied to the applicability of the local consistency algorithm. Thapper and Živný in [98] have shown that this notion can be lifted to general constraint optimisation as well. The algorithmic counterpart to the local consistency algorithm in this setting is the *Sherali-Adams relaxation hierarchy*.

Recall from Section 2.4.1 that every instance I of VCSP(Γ) can be expressed as a 0–1 linear program, and that the *basic linear program relaxation* BLP(I) is obtained from it by simply dropping all integrality constraints on the variables. Solving the (now rational) linear program BLP(I) can be done in polynomial time, and the optimal value obtained in BLP(I) is either equal to the optimal value of I, or less. Hence, the basic LP relaxation essentially gives us an approximation algorithm to solve VCSP(Γ). A common technique to improve the quality of approximation of linear program relaxations is to introduce additional constraints that cut away solutions that are infeasible in the original 0–1 program. The Sherali-Adams relaxation hierarchy [94] defines these additional constraints in a systematic way. Applied to an instance I of VCSP(Γ), we define the (k, l)-level Sherali-Adams relaxation of I, $\mathrm{SA}_{k,l}(I)$, as the following linear program.

Definition 2.67. Let $I = (V, C)$ be an instance of VCSP(Γ) over a domain D, and let $1 \leq k \leq l \leq |V|$. Without loss of generality we can assume that every tuple

$x \in V^i$ for $1 \leq i \leq l$ appears as the scope of some constraint $(x, f, w) \in C$ (otherwise we might simply add a "dummy" constraint with zero weight). Then, $\mathrm{SA}_{k,l}(I)$ is defined as the following linear program.

We have a variable $\lambda_{c,x}$ for every constraint $c = (s, f, w) \in C$, and $x \in D^{\mathrm{ar}(f)}$. The objective is to minimise

$$\min \sum_{\substack{c \in C; \\ c=(s,f,w)}} \sum_{x \in D^{\mathrm{ar}(f)}} \lambda_{c,x} \cdot w \cdot f(x) \tag{2.5}$$

subject to the following constraints.

For each pair of constraints $c, d \in C$, $c = (s, f, w)$ and $d = (t, g, r)$, where the scope t is a projection of s, i.e. $t = \pi_S(s)$ for some subset $S \subseteq [\mathrm{ar}(f)]$, and $\mathrm{ar}(g) \leq k$, and for each $y \in D^{\mathrm{ar}(g)}$,

$$\sum_{\substack{x \in D^{\mathrm{ar}(f)}; \\ \pi_S(x)=y}} \lambda_{c,x} = \lambda_{d,y}. \tag{2.6}$$

Furthermore, for each constraint $c = (s, f, w) \in C$,

$$\sum_{x \in D^{\mathrm{ar}(f)}} \lambda_{c,x} = 1. \tag{2.7}$$

For every constraint $c = (s, f, w)$ and every tuple $t \in D^{\mathrm{ar}(f)}$ with $t \notin \mathrm{Feas}(f)$,

$$\lambda_{c,t} = 0. \tag{2.8}$$

Finally, on every variable $\lambda_{c,x}$, we enforce non-negativity by

$$\lambda_{c,x} \geq 0. \tag{2.9}$$

After having defined $\mathrm{SA}_{k,l}(I)$ formally, let us give some intuition. Intuitively, each variable $\lambda_{c,x}$ determines for a constraint $c = (s, f, w)$ and a tuple $x \in D^{\mathrm{ar}(f)}$ whether the scope s should be assigned to x. An assignment for the LP that sets $\lambda_{c,x}$ to one corresponds to an assignment h for the VCSP, such that $h(s) = x$. As the LP allows rational values to be assigned, its solution typically represents some sort of linear combination of assignments to the CSP. The constraints 2.6 to 2.8 then represent different consistency conditions on the solution. For instance, equation 2.7 ensures that for a fixed constraint $c = (s, f, w)$, the variables $\lambda_{c,x}$ together encode a convex combination of assignments for the scope s, by enforcing that their sum equals to one. Equation 2.8 makes sure that only feasible assignments are considered. Finally, equation 2.9 can be seen as a kind of consistency condition similar to local consistency in the satisfaction setting. It imposes that the assignment is consistent with respect to subsets of sizes up to k: If two scopes s and t share a subset of variables up to size k, then the (distribution of) assignments to s and t on that subset must be the same.

We can now define the notion of width for constraint optimisation problems.

Definition 2.68. A constraint language Γ has width (k, l), if for every instance I of VCSP(Γ), the optimum value of $\mathrm{SA}_{k,l}(I)$ is exactly the optimal value of I. We say Γ has *bounded width*, if k and l are bounded by constants.

Theorem 2.69 ([98])**.** *Let Γ be a constraint language. The following are equivalent:*

(i) VCSP(Γ) *has bounded width.*

(ii) VCSP(Γ) *has width* $(2, 3)$.

(iii) $\mathrm{Pol}^+(\Gamma)$ *satisfies the BWC.*

One of the main contributions of this thesis is to establish a logical characterisation of bounded width VCSPs, and an analogue to Theorem 2.66 for the optimisation setting. This will be done in Chapter 4.

The characterisation of constraint problems in terms of their width has been an important step in the quest of finding and categorising algorithmically accessible constraint problems. Still, it does not fully resolve the dichotomy conjecture, as we know of tractable CSPs that have unbounded width. However, we like to note that for *finite-valued* constraint languages, this gap disappears. Thapper and Živný [96] confirmed that the complexity of finite-valued constraint languages follows a dichotomy.

Theorem 2.70 ([96])**.** *Let Γ be a finite-valued constraint language. Then, either*

- *every instance I of* VCSP(Γ) *is either solved by* BLP(I)*;*
- *or* MAXCUT *reduces in polynomial time to* VCSP(Γ).

As a corrollary, we obtain that VCSP(Γ) for finite-valued Γ can either be solved in polynomial time, or is NP-hard. Related to this, we show in Chapter 5 that the hard cases in this setting can not be solved by sublinear levels of the Lasserre relaxation hierarchy, regardless of any complexity assumptions on P or NP.

2.6.3 Optimisation and satisfaction

The study of constraint problems initially started in the constraint satisfaction setting, and it is in this framework that the Feder-Vardi dichotomy conjecture was formulated. Since then, several sufficient conditions for tractability have been found (such as the bounded width condition), as well as some necessary conditions (see [25, 81]). However, as the status of the conjecture is still unresolved (see Remark 2.9), we do not know of any condition that is proven to be sufficient and necessary at the same time.

Promising progress however has came from the algebraic approach to CSPs. In particular, it turns out that all currently known hardness results are concisely subsumed by a simple algebraic property of the constraint language. Furthermore,

this property is also consistent with all known tractable cases of CSPs, hence it is a strong candidate for the boundary of the conjectured complexity dichotomy. We use the formulation from [76].

Definition 2.71. A k-ary ($k \geq 2$) operation $f : D^k \to D$ is called *cyclic*, if for all $x_1, \ldots, x_k \in D$, it satisfies the identity

$$f(x_1, x_2, \ldots, x_k) = f(x_2, \ldots, x_k, x_1).$$

Theorem 2.72 ([78])**.** *Let* Γ *be a relational constraint language and a rigid core. If* $\mathrm{Pol}(\Gamma)$ *does not contain a cyclic operation of arity at least* 2*, then* $\mathrm{CSP}(\Gamma)$ *is* NP*-hard.*

While the converse of this statement has not been proven, there are also no counter-examples known, and all relational constraint languages Γ currently known to be tractable do have a cyclic operation of arity at least 2. For instance, it is easy to check that any weak near-unanimity operation is also cyclic (see Definition 2.63), and hence bounded width CSPs are an example that is consistent with the theorem. This has lead to the formulation of the so-called *algebraic dichotomy conjecture.*

Conjecture 2.73 ([78])**.** *Let* Γ *be a relational constraint language and a rigid core. If* $\mathrm{Pol}(\Gamma)$ *contains a cyclic operation of arity at least* 2*, then* $\mathrm{CSP}(\Gamma)$ *is tractable; otherwise it is* NP*-hard.*

It is natural to ask whether a similar dichotomy can be conjectured about the optimisation case. As laid out in the previous section, in the special case of finite-valued constraint languages, we know that such a dichotomy exists, and the boundary is described by the bounded width condition. In the more general case, it has been shown [76] that the hardness result from the CSP case carries over.

Theorem 2.74 ([78])**.** *Let* Γ *be a (valued) constraint language and a rigid core. If* $\mathrm{Pol}^+(\Gamma)$ *does not contain a cyclic operation of arity at least 2, then* $\mathrm{VCSP}(\Gamma)$ *is* NP*-hard.*

Even more, it turns out that the complexity of $\mathrm{VCSP}(\Gamma)$ is closely tied to the complexity of its associated *feasibility problem.* For any (valued) constraint language Γ, we define $\mathrm{Feas}(\Gamma) := \{\mathrm{Feas}(f) \mid f \in \Gamma\}$ to be the relational constraint language obtained by taking the feasibility relations of all functions in Γ. The feasibility problem to Γ is then $\mathrm{CSP}(\mathrm{Feas}(\Gamma))$. More precisely, it is shown that $\mathrm{VCSP}(\Gamma)$ is tractable if and only if $\mathrm{CSP}(\mathrm{Feas}(\Gamma))$ is tractable *and* $\mathrm{Pol}^+(\Gamma)$ contains a cyclic operation of arity at least 2.

Theorem 2.75 ([76])**.** *Let* Γ *be a constraint language and a rigid core. If the following conditions hold then* $\mathrm{VCSP}(\Gamma)$ *is tractable:*

- $\mathrm{Pol}^+(\Gamma)$ *contains a cyclic operation of arity at least 2;*
- $\mathrm{CSP}(\mathrm{Feas}(\Gamma))$ *is tractable.*

Otherwise, VCSP(Γ) *is* NP*-hard.*

As a consequence, if the algebraic dichotomy conjecture for the classical CSP case would hold, then we would immediately obtain a corresponding dichotomy also for the general optimisation case.

A crucial part in obtaining the above results are reductions between the optimisation problem VCSP(Γ) and its related satisfaction problems. Some of these reductions are also useful for our purposes, and we state them here.

First we define the *optimality relation* of a function $f : D^m \to \mathbb{Q}_\infty$.

Definition 2.76. Let $f : D^m \to \mathbb{Q}_\infty$ be an m-ary function over D. We associate with it its *optimality relation*

$$\mathrm{Opt}(f) := \{x \in D^m \mid \forall y : f(x) \leq f(y)\}.$$

In words, $\mathrm{Opt}(f)$ is the relation that contains all those tuples x, such that $f(x)$ is the minimum value obtained by f. We write $\mathrm{Opt}(\Gamma) = \{\mathrm{Opt}(f) \mid f \in \Gamma\}$ for constraint language Γ.

Lemma 2.77 ([51])**.** *Let* Γ *be a constraint language, and let* $\Delta := \Gamma \cup \mathrm{Opt}(\Gamma)$*. Then,* VCSP($\Delta$) *is reducible in polynomial time to* VCSP(Γ)*.*

Lemma 2.78 ([78])**.** *Let* Γ *be a constraint language. Then,*

$$\mathrm{Inv}(\mathrm{Pol}^+(\Gamma)) \subseteq \langle \Gamma \cup \mathrm{Opt}(\Gamma) \rangle.$$

Corollary 2.79. *Let* Γ *be a constraint language, and let* $\Delta \subseteq \mathrm{Inv}(\mathrm{Pol}^+(\Gamma))$*. Then,* CSP($\Delta$) *is reducible in polynomial time to* VCSP(Γ)*.*

In Chapter 4 we will show that the polynomial time reductions above can also be formulated in fixed-point logic with counting. This will be helpful when proving our definability results for VCSPs.

Chapter 3

Definability of Boolean constraint satisfaction

Amongst the earliest complexity results on constraint problems was Schaefer's dichotomy theorem [92], which gave a full characterisation of the computational complexity of *Boolean* constraint satisfaction problems, namely CSPs where the domain is fixed to $\{0, 1\}$. In his classification, Schaefer provides a list of six sufficient conditions for a Boolean constraint language Γ that lead to tractability of CSP(Γ), and proves NP-completeness in all other cases. In modern terms, these conditions can be reformulated in terms of polymorphisms of Γ, which tie in with a number of generalisations of Schaefer's result to larger domains, such as [20, 21, 11].

In this chapter we revisit the study of Boolean CSPs from a descriptive point of view. We show that in the Boolean case, the class of tractable CSPs are characterised by definability in *fixed-point logic with rank* (FPR), and conjecture the same for the general case. In particular, we prove the following.

Theorem 3.1. *Let Γ be a relational constraint language over the domain $\{0, 1\}$.* CSP(Γ) *is either polynomial-time tractable and definable in* FPR*; or it is* NP*-complete.*

One of the central open problems in descriptive complexity theory is to determine whether there exists a logic that captures P on all finite structures. The result stated here gives a positive answer to the question in a special case. Namely on the class of Boolean CSPs, the logic FPR does capture all polynomial-time decidable properties. It remains open whether our result can be generalised to CSPs with larger domains, which would be a significant step towards a logic for P.

The techniques used in this chapter are not of particular depth, but illustrate well how essential concepts like definability, logical interpretations, and reductions are used in the remainder of this thesis.

3.1 Schaefer's dichotomy

We restate Schaefer's theorem which characterises all Boolean constraint languages Γ for which CSP(Γ) is tractable. The condition for tractability is stated in terms of polymorphisms of Γ, and the following operations are relevant: The constant operation 0 refers to a unary operation $f_0 : \{0,1\} \to \{0,1\}$ with $f_0(0) = f_0(1) = 0$. Similarly, the constant operation 1 refers to a unary operation defined as $f_1(0) = f_1(1) = 1$. The operations min and max are defined as the usual binary minimum and maximum operations. The majority operation is a ternary operation defined as $\mathrm{maj}(x,y,z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$, and the minority operation is the ternary operation $\mathrm{mnr}(x,y,z) = x \oplus y \oplus z$, where $\oplus$ denotes the Boolean XOR (exclusive OR) operation.

Theorem 3.2 ([92])**.** *Let* Γ *be a Boolean relational constraint language.* CSP(Γ) *is polynomial time tractable if* Pol(Γ) *contains any one of*

- *the constant operations* 0 *and* 1*,*
- *or the minimum and maximum operations* min *and* max*,*
- *or the ternary majority operation* maj*,*
- *or the ternary minority operation* mnr*.*

Otherwise it is NP*-complete.*

Expressing the conditions for tractability in terms of polymorphisms allows us to connect Schaefer's result to the broader algebraic theory behind CSPs. In fact, five of the six tractable cases fall into the class of *bounded width* CSPs (see Definition 2.62). This is due to the following observations.

First, observe that when Γ contains the constant operation 0, all instances of CSP(Γ) are trivially solvable by assigning all variables to 0, and the same is true for the constant operation 1. In these cases, CSP(Γ) clearly has bounded width.

Moreover, the operations min and max belong to the well studied class of *semilattice* operations, while the majority operation is a so-called *near-unanimity* operation. It is known that constraint languages with semilattice operations or near-unanimity operations as polymorphisms satisfy the *bounded width condition* (see Definition 2.64).

Definition 3.3. A binary operation $f : D^2 \to D$ is called a *semilattice* operation if for all $x, y \in D$, it satisfies

$$f(x,y) = f(y,x) \quad \text{and} \quad f(f(x,y),x) = f(x,y).$$

Definition 3.4. A k-ary ($k \geq 2$) operation $f : D^k \to D$ is called a *k-near-unanimity* operation, if for all $x, y \in D$, it satisfies the identities

$$f(y,x,\ldots,x) = f(x,y,x,\ldots,x) = \ldots = f(x,\ldots,x,y) = x.$$

Proposition 3.5 ([12]). *For any constraint language* Γ*, if* $\mathrm{Pol}(\Gamma)$ *contains any semilattice operation or near-unanimity operation of arity* $k \geq 3$*, then it satisfies the bounded width condition.*

As we see, in five of the six tractable cases in Schaefer's classification, the corresponding CSP is of bounded width. Bounded width CSPs are characterised by their definability in Datalog (see Theorem 2.65). Since FPR is strictly more expressive than Datalog, this already gives us a partial proof of Theorem 3.1, namely for exactly these five cases. It remains to show that constraint languages that contain the minority operation are also definable in FPR. As one can guess, it is exactly this case that can not be expressed in Datalog, and we need the added expressive power of FPR.

3.2 Minority and linear equation systems

Already in his original proof, Schaefer noticed that one class of tractable Boolean CSPs essentially represents systems of linear equations over the two-element field. These problems are then easily solved using Gaussian elimination. In the modern terminology of polymorphisms and algebras, these cases turn out to be exactly when the minority operation is a polymorphism of Γ.

Systems of linear equations are a prominent example for a tractable CSP that is not solved by local consistency methods (see Example 2.61), and hence can not be expressed in Datalog, or even $\mathcal{C}^\omega$ (see Theorem 2.66). This has sparked interest in finding logics that *are* powerful enough to express solvability of linear equations. One such logic is FPR (see Section 2.5.3), which is the extension of inflationary fixed-point logic with a *rank* operator that defines the rank of a given matrix.

Given the equivalence between Boolean constraint languages Γ with a minority polymorphism and systems of linear equations, it is not difficult to obtain a corresponding definability result for $\mathrm{CSP}(\Gamma)$. If we show that instances of $\mathrm{CSP}(\Gamma)$ can be interpreted in a suitable logic as systems of linear equations, and the solvability of those are definable in FPR, then it follows that $\mathrm{CSP}(\Gamma)$ is also definable in FPR. The following propositions establish this argument.

We represent systems of linear equations over the Boolean field as conjunctions of clauses of the form $(x \oplus y \oplus z = 0)$ and $(x \oplus y \oplus z = 1)$, since the Boolean XOR operator $\oplus$ is also exactly the addition operator modulo 2. The decision problem of whether or not a given system is solvable is denoted by 3-LIN. In terms of relational structures, we define $\tau_{\mathrm{3LIN}} := \{E_0, E_1\}$ as the vocabulary of 3-LIN instances, where the ternary relation E_i encodes equations of the kind $(x \oplus y \oplus z = i)$.

Proposition 3.6 ([92]). *Let* R *be an* m*-ary relation over* $\{0, 1\}$ *that has* mnr *as a polymorphism. There is a conjunction of linear equations* ψ *such that* $x \in R \Leftrightarrow \psi(x)$*.*

Proposition 3.7. *Let* Γ *be Boolean relational constraint language that has* mnr *as a polymorphism. Then* $\mathrm{CSP}(\Gamma) \leq_{\mathrm{FO}}$ 3-LIN*.*

Proof. Let $\mathbf{I} = (V, (R^{\mathbf{I}})_{R\in\Gamma})$ be an instance of CSP(Γ) as a relational structure. According to Proposition 3.6, each $R^{\mathbf{I}}$ is equivalent to an instance ψ_R of 3-LIN. Hence, the instance $\mathbf{I}$ is satisfiable if and only if the conjunction of linear equations

$$\bigwedge_{R\in\Gamma}\bigwedge_{t\in R^{\mathbf{I}}}\psi_R(t)$$

is solvable.

This 3-LIN instance is definable in first-order logic from $\mathbf{I}$, by the following FO-interpretation. The universes are the same, and the relations E_0 and E_1 are defined as follows. We write $E_i^{\psi_R}$ for the relation that contains those triples (x, y, z) such that the equation $(x \oplus y \oplus z = i)$ appears in ψ_R.

$$\phi_{E_0}(x, y, z) = \bigvee_{R\in\Gamma} E_0^{\psi_R}(x, y, z)$$
$$\phi_{E_1}(x, y, z) = \bigvee_{R\in\Gamma} E_1^{\psi_R}(x, y, z).$$

□

Proposition 3.8 ([38])**.** 3-LIN *is definable in* FPR*.*

As an immediate consequence of Propositions 3.7 and 3.8, we obtain the definability result for CSP(Γ).

Corollary 3.9. *Let* Γ *be Boolean relational constraint language that has* mnr *as a polymorphism. Then* CSP(Γ) *is definable in* FPR*.*

Together with Schaefer's result and the fact of the other five tractable cases identified by Schaefer are definable in Datalog, which is a fragment of FPR, this implies the dichotomy stated in Theorem 3.1.

3.3 Beyond the Boolean case

The famous dichotomy conjecture posed by Feder and Vardi claims that the dichotomy for Boolean CSPs also extends to CSPs of arbitrary domains. Since then, Schaefer's result has in fact been successfully generalised to a number cases, including for CSPs over three-element domains [20]. Instrumental to these advancements has been the development of the algebraic approach to CSPs, and in particular a class of polynomial time algorithms that solve CSPs that are not solvable by local consistency methods. This type of algorithm can be seen as an extension of Gaussian elimination, and correspondingly, these algorithms are suitable for constraint languages that have a type of polymorphism that generalises the minority operation. The first of such algorithms was found by Bulatov [19, 24], which solves CSP(Γ) when Γ has a so-called *Mal'tsev* polymorphism.

Definition 3.10. A ternary operation $f : D^3 \to D$ is called a *Mal'tsev* operation, if for all $x, y \in D$, it satisfies the identity

$$f(x, y, y) = f(y, y, x) = x.$$

Proposition 3.11 ([19])**.** *Let* Γ *be a relational constraint language that has a Mal'tsev polymorphism. Then,* $\mathrm{CSP}(\Gamma)$ *is polynomial time tractable.*

Further developments have resulted in the *few subpowers* algorithm [33] which solves a larger class of CSPs, characterised again in terms of polymorphisms.

Definition 3.12. A k-ary ($k \geq 2$) operation $f : D^k \to D$ is called a *k-edge* operation, if for all $x, y \in D$, it satisfies the identities

$$f(y, y, x, \ldots, x) = f(y, x, y, x, x, \ldots, x) = x$$

and

$$f(x, x, x, y, \ldots, x) = f(x, x, x, x, y, \ldots, x) = \ldots = f(x, x, \ldots, y) = x.$$

Proposition 3.13 ([66, 12])**.** *Let* Γ *be a relational constraint language that has a k-edge operation as a polymorphism for some $k \geq 2$. Then,* $\mathrm{CSP}(\Gamma)$ *is polynomial time tractable.*

Observe that constraint languages containing a Mal'tsev operation as polymorphism also have a 2-edge polymorphism: An operation $f(x, y, z)$ is Mal'tsev if and only if $f(y, x, z)$ is a 2-edge operation. Similarly, the ternary minority operation over any domain is in fact a Mal'tsev operation. More interestingly, in the sepcial case of the Boolean domain, the converse also holds in some sense: Every Boolean constraint language that has a Mal'tsev polymorphism also has mnr as a polymorphism. Due to the defining identities of Mal'tsev operations, there are only four distinct Mal'tsev operations g over the Boolean domain. A case distinction reveals that one of the four cases is exactly mnr, and in the other three it holds that $\mathrm{mnr}(x, y, z) = g(g(y, x, z), y, g(x, z, y))$. Hence, Schaefer's result shows that the tractable Boolean CSPs are in fact either bounded width, or have a Mal'tsev polymorphism. This pattern holds up even today, as all known tractable CSPs are either solvable via local consistency methods, or the few subpowers algorithm, or some combination of both.

From a definability point of view, the state of knowledge is rather incomplete. While there exists a number of results connecting bounded width CSPs to logic (in particular Datalog, see Theorem 2.65), little is known about the class of CSPs that are tractable via few subpowers. The result presented in this chapter is a first step in this direction, as it establishes a connection between tractability via few subpowers (languages with Mal'tsev/minority polymorphisms) and definability in FPR. While this is a fairly direct consequence of the specific fact that Boolean CSPs with a minority polymorphism are equivalent to linear equation systems, there

is hope that this can be generalised to larger domains as well. A potential direction could be to study how exactly the few subpowers algorithm is related to Gaussian elimination, and how CSPs with Mal'tsev or k-edge polymorphisms relate to systems of equations. We leave this as an open question.

Conjecture 3.14. *Let* Γ *be Boolean relational constraint language that has a* k*-edge operation as a polymorphism for some* $k \geq 2$*. Then* $\mathrm{CSP}(\Gamma)$ *is definable in* FPR*.*

Chapter 4

Definability of constraint optimisation

This chapter contains material published in [40].

In the previous chapter we showed a dichotomy for a very restricted class of constraint problems. Namely we assumed the domain to be Boolean, and we only considered constraint *satisfaction* problems, without any optimisation component. In this chapter, we study the general constraint framework, where both of these restrictions are lifted. This means that we consider general *valued* CSPs over arbitrary domains. We show that a definability dichotomy exists again in this general framework, however this time separating the logics FPC and $\mathcal{C}^\omega$. In particular, we show that any VCSP(Γ) is either definable in FPC – or it is not definable in the stronger infinitary logic $\mathcal{C}^\omega$.

This result can be seen as an extension of several existing works. As one of the first definability results on CSPs, it was established early by Feder and Vardi [48] that all bounded-width CSPs are definable in Datalog. Since then, the class of bounded-width CSPs has received much attention, with several seminal results coming from algebra. In particular, a classification of the algebraic properties of Γ in terms of varieties of finite algebras from tame congruence theory [64] has proved useful. In this context, Larose and Zádori [81] showed that if the variety generated by Alg(Γ) admits the unary or affine type, then CSP(Γ) is not definable in Datalog, and conjectured the converse. Later, Atserias, Bulatov, and Dawar [5] strengthened this result, showing that those undefinable CSPs even remain undefinable in the stronger logic $\mathcal{C}^\omega$. This result was partly obtained by improving previously known polynomial-time reductions to Datalog-reductions. A similar approach is also part of our proof strategy, as we translate certain known polynomial-time reductions to reductions in the logic FPC. Even later, Barto and Kozik [10] achieved a full algebraic characterisation of the bounded-width case, affirming the previously conjectured converse of the Larose-Zádori result. In their follow-up work [9], the algebraic condition for bounded width was then shown to be equivalent to a condition now known as the *bounded width condition* (BWC), as defined in Definition 2.64. These results

together imply the definability dichotomy for classical constraint satisfaction (see Theorem 2.66): Any CSP has either bounded width and is definable in Datalog; or it is not definable in $\mathcal{C}^\omega$. Our result here lifts this dichotomy to the realm of VCSPs. And as in the satisfaction case, the condition for definability is the notion of bounded width, generalised for VCSPs (see Definition 2.68).

The notion of width for VCSPs has been first introduced by Thapper and Živný [98], where they show that the algebraic characterisation of width can be meaningfully extended to optimisation problems as well. They also provide a polynomial time algorithm for solving all bounded width VCSPs based on linear programming. In fact, it turns out the third level of the Sherali-Adams relaxation hierarchy is sufficient to solve any bounded width VCSP. As in the satisfaction case the notion of bounded width does not describe a dichotomy for complexity classes: There are problems that do not have bounded width, but are still solvable in polynomial time.

In the context of VCSPs and definability, fixed-point logic with counting (FPC) plays an important role. As VCSPs are optimisation problems, it is often necessary to express cardinalities and numbers, which can be realised using counting operators in logic. The logic FPC is a natural candidate, as extending Datalog with a counting operator results in exactly the expressive power of FPC [52]. This is reflected in our dichotomy result for bounded width VCSPs. Our result gives a descriptive perspective on the notion of bounded width in the valued setting, and establishes the following dichotomy: We show that all VCSPs satisfying the bounded width condition are definable in FPC, while all other VCSPs are not definable in $\mathcal{C}^\omega$.

Note that in the special case of *finite-valued* CSPs, namely the class of "pure" optimisation problems, there is a dichotomy also in computational complexity terms. In this framework, Thapper and Živný [96] were able to show that all those VCSP that are not solvable by the linear programming approach are a reduction target for the NP-complete MAXCUT problem. As our result shows a dichotomy for the general case, it also directly implies the same dichotomy for the special case. In the case of finite-valued CSPs, the worlds of complexity and definability are aligned: The bounded width cases are solvable in polynomial time and definable in FPC – and all other cases are NP-complete and not definable in $\mathcal{C}^\omega$. Formally we prove the following theorem.

Theorem 4.1. *For any constraint language* Γ*,*

- *either* $\mathrm{Pol}^+(\Gamma)$ *satisfies the* bounded width condition*, and* $\mathrm{VCSP}(\Gamma)$ *is definable in* FPC*,*
- *or* $\mathrm{VCSP}(\Gamma)$ *is not definable in* $\mathcal{C}^\omega$*.*

Recall the definition of the bounded width condition from Definition 2.64.

The rest of this chapter is structured as follows. We start in Section 4.1 by putting the theory of polymorphisms of constraint languages into a descriptive perspective. That is, we show that not only do the algebraic properties of $\mathrm{Pol}(\Gamma)$ or

wPol(Γ) determine the computational complexity of the language, but they are also deeply connected to its definability. In particular, we translate many of the classical polynomial-time reductions between constraint problems into first-order or FPC reductions. This allows us to use the power of the algebraic approach to prove (un)definablity results. Then, in Section 4.2, we show that certain linear programs associated with VCSPs can be defined via FPC-interpretations. This opens up the path to use the definability result for LPs from [3, 4] (Theorem 2.49) for constraint optimisation problems. Finally, the two parts of the dichotomy are then shown in the two remaining sections. In Section 4.3 we show that bounded width VCSPs are definable in FPC, while in Section 4.4 we provide the undefinability part of the dichotomy, by lifting the undefinability result for relational CSPs [5] (Theorem 2.66) to the general valued case.

4.1 Definable reductions

An essential part of the machinery that led to recent complexity results on constraint problems is that the computational complexity of VCSP(Γ) is robust under certain changes to Γ. In other words, closing the constraint language Γ under certain natural operations does not change the complexity of the problem. This is established by showing that the distinct problems obtained are inter-reducible under polynomial-time reductions. We have listed some of these reductions in Sections 2.3 and 2.6.

In this section we show that these reductions can be expressed as interpretations in a suitable logic (in some cases first-order logic suffices, and in others we need the power of counting). In particular, this means that these changes to the constraint language are not only robust from a computational complexity perspective, but also from a descriptive point of view.

The following lemma summarises the reductions shown between distinct VCSPs. Note that we treat VCSP(Γ) as a decision problem: An input consists of an instance $I = (V, C)$ and a rational threshold t, and the task is to decide whether the optimum value of I is less than or equal to t. Recall that $\langle\Gamma\rangle$ denotes the closure of Γ under the operations of *expression*, *scaling*, and *translation* by constants (see Definition 2.19). Furthermore, we use Γ_c for the *rigid core* obtained from the core of Γ by adding all singleton relations (Definition 2.59), and Opt(Γ) is the *set of optimality relations* of Γ (Definition 2.71).

Lemma 4.2. *Let* Γ *and* Δ *be two constraint languages over some domain* D*. Then,*

(i) if $\Delta \subseteq \langle\Gamma\rangle$*, then* $\mathrm{VCSP}(\Delta) \leq_{\mathrm{FPC}} \mathrm{VCSP}(\Gamma)$*;*

(ii) if Γ *is a core of* Δ*, then* $\mathrm{VCSP}(\Delta) \leq_{\mathrm{FO}} \mathrm{VCSP}(\Gamma)$*;*

(iii) if Γ *is a core, then* $\mathrm{VCSP}(\Gamma_c) \leq_{\mathrm{FPC}} \mathrm{VCSP}(\Gamma)$*;*

(iv) if $\Delta \subseteq \Gamma \cup \mathrm{Opt}(\Gamma)$*, then* $\mathrm{VCSP}(\Delta) \leq_{\mathrm{FPC}} \mathrm{VCSP}(\Gamma)$*.*

Furthermore, we show that the complexity of classical CSPs and VCSPs are connected via universal algebra. Recall Section 2.3.1 for definitions.

Lemma 4.3. *Let Γ be a constraint language and let $\Delta \subseteq \mathrm{Inv}(\mathrm{Pol}^+(\Gamma))$. Then,*

$$\mathrm{CSP}(\Delta) \leq_{\mathrm{FPC}} \mathrm{VCSP}(\Gamma).$$

In the following sections we provide proofs of the above results.

Expressibility, scaling, and translation

We aim to prove Lemma 4.2.i. That is, we show that extending a constraint language Γ by functions that are expressible in Γ, or are of the form $f = a \cdot g + b$ for $g \in \Gamma, a, b \in \mathbb{Z}, a > 0$, does not change whether $\mathrm{VCSP}(\Gamma)$ is definable in FPC.

Proof of Lemma 4.2.i. Note that we can assume that Δ contains at least one function not in Γ, as otherwise any $\mathrm{VCSP}(\Delta)$ instance can be interpreted directly as a $\mathrm{VCSP}(\Gamma)$ instance.

We will show that extending a constraint language Γ by functions that are expressible in Γ preserves the FPC-definability of $\mathrm{VCSP}(\Gamma)$. Our construction here is similar to the polynomial time reduction proved in [31], however ensuring it is definable in FPC. In a second step, we show that adding functions to Γ obtained by scaling and translation preserves definability as well. The reductions here are optimum-preserving.

Let Γ be a constraint language over a domain D, $f : D^m \to \mathbb{Q}_\infty$ a function that is expressible in Γ, and $\Delta := \Gamma \cup \{f\}$. Consider an instance $I = (V, C)$ of $\mathrm{VCSP}(\Delta)$. Since f is expressible in Γ, there is an instance $I_f = (V_f, C_f)$ of $\mathrm{VCSP}(\Gamma)$, as well as a list of variables $u = (u_1, \ldots, u_m) \in V_f^m$ that witness the expressibility of f according to Definition 2.19. Recall now that $f(x_1, \ldots, x_m)$ can be defined as the minimum value obtained by I_f when the variables $u_1, \ldots, u_m$ are fixed to $x_1, \ldots, x_m$. Hence, the idea is to replace each constraint in C of the form (s, f, q) by a copy of the instance I_f, where we substitute the variables $u_1, \ldots, u_m$ by the variables in the scope $s_1, \ldots, s_m$. In this way, we can eliminate all occurences of f in the constraints, and obtain in the end an instance of $\mathrm{VCSP}(\Gamma)$.

Formally, we aim to define an instance $J = (U, E)$ of $\mathrm{VCSP}(\Gamma)$ that has the same optimal solution as I. The set of variables U consists of the variables in V plus a fresh copy of the variables in V_f for each constraint in C that uses the function f.

$$U = V \mathbin{\dot\cup} \{(v, c) \mid \forall c \in C, c = (s, f, q), v \in V_f\}. \tag{4.1}$$

Each constraint $c = (s, f, q) \in C$ gives rise to a set of constraints E_c, representing a copy of the constraints in C_f.

$$E_{(s,f,q)} = \{(t_{u \mapsto s}, g, q \cdot r) \mid \forall (t, g, r) \in C_f\}, \tag{4.2}$$

where $t_{u\mapsto s}$ denotes the tuple t where each occurence of a variable in $u_1, \dots, u_m$ is replaced by the corresponding variable s_i. The set of constraints E is then simply the union of all sets E_c with the remaining constraints in C that do not contain f.

$$E = \{c \in C \mid c = (s, g, q), g \neq f\} \cup \bigcup_{c \in C} E_c. \tag{4.3}$$

We now give an FPC-interpretation that defines (a relational representation of) the instance $J = (U, E)$ given (a relational representation of) an instance $I = (V, C)$ of VCSP(Δ). Recall that instances of VCSP(Γ) and VCSP(Δ) are encoded as structures of the vocabularies $\tau_\Gamma = (<, (R_f)_{f\in\Gamma}, W)$ and $\tau_\Delta = (<, (R_f)_{f\in\Delta}, W)$ respectively. We aim to define an FPC reduction $\Theta = (\delta, \varepsilon, \phi_<, (\phi_{R_f})_{f\in\Gamma}, \phi_W)$ such that $\mathbf{J} = \Theta(\mathbf{I})$ corresponds to the above construction of the instance J.

Let $\mathbf{I}$ be an encoding of an instance $I = (V, C)$ of VCSP(Δ) with a three-sorted universe $\mathrm{dom}(\mathbf{I}) = V \dot\cup C \dot\cup B$. Furthermore we associate with the function f a fixed instance $I_f = (V_f, C_f)$, and a fixed tuple $u = (u_1, \dots, u_m)$, witnessing the expressibility of f. As the witness instance is independent of the input $\mathbf{I}$, we fix an encoding of the sets V_f and C_f as initial segments of the natural numbers $\mathcal{V}_f := \{1, \dots, |V_f|\}$ and $\mathcal{C}_f := \{1, \dots, |C_f|\}$, as well as two bijections $var : V_f \to \mathcal{V}_f$ and $con : C_f \to \mathcal{C}_f$. This is useful as we can then use number terms in our formula to refer to the elements of V_f and C_f.

We define the universe of $\mathbf{J}$ as a three-sorted set $\mathrm{dom}(\mathbf{J}) = U \dot\cup E \dot\cup B'$ consisting of variables U, constraints E, and an initial segment of natural numbers B'. The set U is defined by the formula

$$\delta_U(x, \mu) = (x \in C \wedge \exists y \in V^m : R_f(y, x) \wedge \mu \in \mathcal{V}_f)) \vee (\mu = 0 \wedge x \in V).$$

In other words, the elements of U consist of pairs (x, μ), where $x \in V \cup C$ and μ is a natural number, and we make the following case distinction: Either x represents a constraint $x = (y, f, q)$ in C, and μ encodes a variable in V_f; then the pair represents one of the fresh variables in $C \times V_f$. Or, $x \in V$ is a variable, and $\mu = 0$, and the pair simply represents an element of V. This aligns with the definition of U in (4.1) above.

Similarly, the constraints E are given by

$$\delta_E(x, \mu) = x \in C \wedge (\mu = 0 \vee \exists y \in V^m : R_f(y, x) \wedge \mu \in \mathcal{C}_f).$$

Again, the elements of E are pairs (x, μ), with $x \in C$, and μ an element of the number domain. Here we require that if x refers to a constraint containing f, i.e. it is of the form $x = (y, f, q)$, then μ encodes a constraint in C_f. Otherwise, $\mu = 0$ and x represents one of the remaining constraints in C not containing f. This corresponds to the definition of E in (4.2) and (4.3).

For the domain of bit positions, we just need to make sure that the set is large enough to encode all weights in J. Taking $B' = B^2$ suffices, so

$$\delta_{B'}(x_1, x_2) = x_1, x_2 \in B$$

and we take $\phi_<(x, y)$ to be the formula that defines the lexicographic order on pairs.

The constraints of J are encoded in the relations R_g, $g \in \Gamma$. For a k-ary function g, this is defined by a formula ϕ_{R_g} in the free variables $((x_1, \mu_1), \ldots, (x_k, \mu_k), (e, \nu))$ where each (x_i, μ_i) ranges over elements of U, and (e, ν) ranges over elements of E. To be precise, we define the formula by:

$$\begin{aligned}\phi_{R_g} = (\exists y \in V^m : R_g(y, e) \wedge \nu = 0 \wedge \bigwedge_{1 \le i \le k} (y_i = x_i \wedge \mu_i = 0)) \\ \vee (\exists y \in V^m : R_f(y, e) \wedge \psi),\end{aligned}$$

where

$$\psi = \bigvee_{\substack{e' \in C_f; \\ e' = (t,g,r)}} \left(\nu = \ con(e') \wedge \bigwedge_{i : t_i \in u} (x_i = e \wedge \mu_i = var(t_i)) \wedge \bigwedge_{i : t_i \notin u} (x_i = y_i \wedge \mu_i = 0) \right).$$

Again, we have a case distinction here. If $\nu = 0$, and e encodes a constraint in C that does not contain f, then we simply copy the constraint from the input instance by ensuring that all x_i are original variables of V, and all μ_i are zero. If e however encodes a constraint that does contain f, then the formula ψ comes into play. It ensures that the definition of (4.2) is satisfied.

The weight relation follows a similar case distinction. If (e, ν) encodes a constraint $c \in C$ that does not contain f, then the weight is simply the original weight of c. If (e, ν) encodes a constraint that does contain f, then its weight is the product of the original constraint c in C and the weight of $con^{-1}(\nu)$ in C_f. Since all arithmetic operations on number terms can be realised in FPC, we can define the weight relation ϕ_W in FPC as well. We skip its explicit formulation here, as the main ideas are already present in the previous cases.

We now turn to the operations of scaling and translation. That is, we consider the constraint language $\Delta := \Gamma \cup \{f\}$ where $f = a \cdot g + b$ for some $g \in \Gamma, a, b \in \mathbb{Z}, a > 0$. Note that adding constants to the value of constraints never changes the optimal solution of the instance. Hence, we only need to take care of the scaling factor a. This can be achieved by changing the weights accordingly.

Let $I = (V, C)$ be an instance of VCSP(Δ). We aim to construct an instance $J = (U, E)$ of VCSP(Γ) with the same optimal solution. The set of variables of J stays V, and any constraint in C that does not contain f is carried over unchanged to E. For the other constraints in C of the form (s, f, q), we simply add to E the constraint $(s, g, a \cdot q)$. This construction can be suitably defined as an FPC interpretation, using similar ideas as above. □

Cores and constants

In this section we aim to establish FPC-reductions between the constraint problems over Γ and its rigid core Γ_c. This will allow us to assume the rigid core property for

all constraint languages that we consider later. The proofs to Lemma 4.2.ii and iii are as follows.

Proof of Lemma 4.2.ii. Note that if Γ is a core of a constraint language Δ, then VCSP(Δ) is reducible by a first-order reduction to VCSP(Γ): Since Γ is a core of Δ, the functions in Γ are exactly those in Δ, only restricted to some subset of D. Hence we can interpret any instance of VCSP(Δ) directly as an instance of VCSP(Γ). By Lemma 2.57, the optimum for both instances are the same, and this is already a reduction. □

Proof of Lemma 4.2.iii. Let Γ be a constraint language over a domain D, with $D = \{a_1, \ldots, a_n\}$. Recall that Γ_c is obtained by adding to Γ all unary singleton relations $S_i = \{a_i\}$ for all $a_i \in D$. We aim to show that VCSP(Γ_c) FPC-reduces to VCSP(Γ). The reduction here is a threshold reduction. That is, we aim to construct an FPC-interpretation that maps an instance (I, t) of VCSP(Γ_c) to an instance (J, t') of VCSP(Γ) such that the optimal value of I is less than or equal to t if and only if the optimal value of J is less than or equal to t'. The construction follows the proof for the polynomial time reduction found in [78], we show that this can be realised in FPC.

First, we observe that since Γ is a core, it follows from Lemma 2.58 that there exists an n-ary function $f : D^n \to \mathbb{Q}_\infty$, $f \in \langle\Gamma\rangle$, as well as a positive rational number $p \in \mathbb{Q}^+$, such that the following conditions are true:

- $f(x_1, \ldots, x_n) = 0$ if and only if the unary operation $g : D \to D, g(a_i) = x_i$ belongs to $\mathrm{Pol}^+(\Gamma)$,
- and $f(x_1, \ldots, x_n) > p$ otherwise.

Now, given an instance $I = (V, C)$ of VCSP(Γ_c) and a threshold t, we aim to construct an instance $J = (U, E)$ of VCSP($\langle\Gamma\rangle$) along with a corresponding threshold t'. By Lemma 4.2.i we know that VCSP($\langle\Gamma\rangle$) FPC-reduces to VCSP(Γ), so this suffices to show our claim. We define J and t' as follows.

The set of variables U is obtained from V by adding n additional fresh variables. That is, we define $U := V \dot\cup \{v_1, \ldots, v_n\}$. For every constraint $c \in C$ of the form $c = (x, S_i, q)$, we add the constraint $e = ((x, v_i), R_=, q)$ to the constraint set E, where $R_=$ is the binary equality relation $R_=(x, y) = \{(x, y) \in D^2 \mid x = y\}$. Note that $R_=$ is expressible over every non-trivial constraint language, hence $R_= \in \langle\Gamma\rangle$. Finally, we add to E a new constraint of the form $((v_1, \ldots, v_n), f, K)$, where K is chosen as follows. Let M be defined as $M := \sum_{(x,g,q)\in C} q \cdot \max_{y\in \mathrm{Feas}(g)} g(y)$, i.e. M is an upper bound on the value of I that any feasible (finite cost) solution can obtain. Then, we choose K such that $K \cdot q > M$. Finally, we set the threshold t' as $t' := \min(t, M)$.

To see that this actually is a reduction, we consider the following cases:

1. If J is not feasible, i.e. $\mathrm{Val}_J = \infty$, then I is also infeasible. Suppose otherwise, and let $h_I : V \to D$ be an assignment for I that has finite cost. Then, the

assignment $h_J : U \to D$ defined by $h_J(x) = h_I(x)$ for $x \in V$ and $h_J(v_i) = a_i$ for $v_i \in \{v_1, \ldots, v_n\}$ also has finite cost for J.

2. If $\mathrm{Val}_J \leq M$, then the optimal values for I and J are the same. Let h_J be an optimal solution to J with a value less than or equal to M. Due to the constraint $((v_1, \ldots, v_n), f, K)$ in J, this can only occur if $f(h_J(v_1), \ldots, h_J(v_n)) = 0$, and the operation $g : D \to D, g(a_i) = h_J(v_i)$ belongs to $\mathrm{Pol}^+(\Gamma)$. Since Γ is a core, all operations in $\mathrm{Pol}^+(\Gamma)$ are bijective, and by Lemma 2.56, the assignment $g^{-1} \circ h_J$ is also an optimal assignment for J. Its restriction to V is then also an optimal solution to I.

3. If $\mathrm{Val}_J > M$, then I is infeasible. By the same argument as in the first case, if I had an assignment h_I of finite cost, then the corresponding assignment h_J of J would also have finite cost. Furthermore, its value would be less than or equal to M, since $g(a_i) = h_J(v_i) = a_i$ is the identity and trivially belongs to $\mathrm{Pol}^+(\Gamma)$.

We now formulate the above construction in terms of an FPC-interpretation. Let $I = (V, C)$ be a given instance of $\mathrm{VCSP}(\Gamma_c)$ represented as a τ_{Γ_c} structure $\mathbf{I}$ with a three-sorted universe $\mathrm{dom}(\mathbf{I}) = V \dot\cup C \dot\cup B$. We define an interpretation Θ such that $\mathbf{J} = \Theta(\mathbf{I})$ represents $J = (U, E)$ as described above. The universe of J is again a three-sorted set consisting of variables U, constraints E, and bit positions B'. We define the set of variables by

$$\delta_U(x, \mu) = x \in V \wedge \mu \in \{0, 1, \ldots, n\},$$

where we identify elements according to the equivalence relation

$$\varepsilon_U((x, \mu), (y, \nu)) = (x = y \wedge \mu = \nu = 0) \vee (\mu = \nu \wedge \mu > 0).$$

That is, an element $(v, 0)$ encodes the variable $v \in V$, and any (v, i) with $i > 0$ simply encodes the number i. Together this gives us the desired set $U = V \dot\cup \{1, \ldots, n\}$. Similarly, the set of constraints is defined by

$$\delta_E(x, \mu) = x \in C \wedge \mu \in \{0, 1\},$$

with an equivalence relation

$$\varepsilon_E((x, \mu), (y, \nu)) = (x = y \wedge \mu = \nu = 0) \vee (\mu = \nu = 1).$$

Here, any element $(c, 0)$ encodes a constraint obtained from an original constraint $c \in C$, and any $(c, 1)$ encodes the single distinguished constraint that we add to E. For the bit domain of $\mathbf{J}$, we again simply choose a large enough subset of the natural numbers.

The constraints of J are encoded in the relations $R_g, g \in \Gamma$, and are defined by formulas ϕ_{R_g}. For a k-ary function g, the formula ϕ_g has the free variables

$((x_1, .\mu_1), \ldots, (x_k, \mu_k), (e, \nu))$, where each (x_i, μ_i) is of the variable sort, and (e, ν) is of the constraint sort. For all $g \in \Gamma, g \notin \{R_=, f\}$, we leave the relation essentially untouched from $\mathbf{I}$,

$$\phi_{R_g} := \bigwedge_{1 \leq i \leq k} (\mu_k = 0) \wedge \nu = 0 \wedge R_g(x_1, \ldots, x_k, e).$$

For each constraint in C of the form (x, S_i, q), we add in constraints that use the binary equality relation $R_=$,

$$\phi_{R_=} := \mu_1 = 0 \wedge \mu_2 > 0 \wedge \nu = 0 \wedge \bigvee_{1 \leq i \leq n} \exists y \in V : R_{S_i}(y, e) \wedge x_1 = y \wedge i = \mu_2.$$

Finally, there is the single constraint that uses the function f.

$$\phi_{R_f} := \bigwedge_{1 \leq i \leq k} \mu_i = i \wedge \nu = 1.$$

For the definition of the weight relations and the threshold, we again use the Immerman-Vardi theorem. Since the weights and the threshold are polynomial-time computable from the instance I, it follows that we can define the formulas ϕ_W and $\phi_{t'}$ in FPC, as they operate over the ordered bit domain B'. □

Optimisation and satisfaction

Finally, we show that the definability of the optimisation problem VCSP(Γ) depends on its related satisfaction problem CSP(Δ) with $\Delta \subseteq \mathrm{Inv}(\mathrm{Pol}^+(\Gamma))$. We prove Lemma 4.2.iv and Lemma 4.3.

Proof of Lemma 4.2.iv. For a fixed $f \in \Gamma$, let $\Delta \subseteq \Gamma \cup \{\mathrm{Opt}(f)\}$ be a constraint language. We aim to prove $\mathrm{VCSP}(\Delta) \leq_{\mathrm{FPC}} \mathrm{VCSP}(\Gamma)$. This is done by showing that for any single function $f \in \Gamma$, adding $\mathrm{Opt}(f)$ to the language does not change its definability with regards to FPC. As we assume Δ to be finite, the argument can be iterated to include all of Δ. The construction here is similar to the one in [51], and again our contribution is its formulation in the logic FPC. The reduction here is a threshold reduction, i.e. it does not necessarily preserve the optimal value of the input instance. Instead, given an instance I of VCSP(Δ) and a threshold t, we construct an instance J of VCSP(Γ) and a value t', such that $\mathrm{Val}_I \leq t$ if and only if $\mathrm{Val}_J \leq t'$.

The idea is as follows. Note that we can assume without loss of generality that all functions in Γ are non-negative, and that the minimum value of f is zero (otherwise simply translate functions by positive constants until this is the case). Furthermore, let p be the smallest positive value f can take. For an instance $I = (V, C, t)$ of VCSP(Δ) we also define $M := \sum_{(x,g,q) \in C} q \cdot \max_{y \in \mathrm{Feas}(g)} g(y)$, serving as an upper bound on the largest finite cost any solution of I can attain.

We construct the instance $J = (U, E)$ of VCSP(Γ) to have the same set of variables as I, and we replace any constraint of the form $(x, \mathrm{Opt}(f), q) \in C$ by $(x, f, K \cdot p)$ where we choose $K \cdot p > M$. Now observe that any feasible solution of I is also feasible for J, and the optimal values are the same, since the constraints that were changed contribute a value of zero in both cases. On the other hand, any solution to J with a value less than M, is also a feasible solution to I with the same value. Only solutions to J whose value is larger than M, correspond to infeasible solutions to I. Hence, by setting the threshold $t' := \min(t, M)$ we obtain a reduction from VCSP(Δ) to VCSP(Γ).

In terms of a FPC-reduction, we define an interpretation Θ that maps a structure $(\mathbf{I}, t)$ to a structure $(\mathbf{J}, t')$ according to the above construction. Let $\mathbf{I}$ be the representation of an instance $I = (V, C)$ with the three-sorted universe $\mathrm{dom}(\mathbf{I}) = V \dot\cup C \dot\cup B$. The structure $\mathbf{J} = \Theta(I)$ has a three-sorted universe $\mathrm{dom}(\mathbf{J}) = U \dot\cup E \dot\cup B'$, where the variable set U and the constraint set E are unchanged from $\mathbf{I}$, and the set of bit positions B' is again chosen to be large enough to express all weights. That is, we have

$$\delta_U(x) = x \in V; \quad \delta_E(x) = x \in C; \quad \delta_{B'}(\mu) = \mu \in \{1, \dots, K \cdot p\}.$$

For all functions $g \in \Gamma, g \neq f$, the constraints are carried over from I unchanged. Hence we simply write

$$\phi_{R_g}(x, c) = R_g(x, c).$$

For the distinguished function f, we have to replace the constraints in I that use $\mathrm{Opt}(f)$.

$$\phi_{R_f}(x, c) = R_{\mathrm{Opt}(f)}(x, c).$$

Finally, the weights as well as the threshold are polynomial-time computable from the instance I. Since FPC can define any polynomial-time computable function on an ordered domain, this concludes the proof. □

Proof of Lemma 4.3. Let $\Delta \subseteq \mathrm{Inv}(\mathrm{Pol}^+(\Gamma))$. By Lemma 4.2.ii and iii we can assume that Δ is a rigid core. By Lemma 2.78, we have $\Delta \subseteq \langle \Gamma \cup \mathrm{Opt}(\Gamma) \rangle$. It follows that $\mathrm{CSP}(\Delta) \leq_{\mathrm{FPC}} \mathrm{VCSP}(\langle \Gamma \cup \mathrm{Opt}(\Gamma) \rangle)$. We can then apply Lemma 4.2.i and iv to reduce $\mathrm{VCSP}(\langle \Gamma \cup \mathrm{Opt}(\Gamma) \rangle) \leq_{\mathrm{FPC}} \mathrm{VCSP}(\Gamma)$, and obtain the claimed reduction by composition. □

4.2 VCSPs and linear programming

One of the driving forces behind many complexity results for VCSPs has been its connection to linear programming. Since VCSPs can be represented as 0–1 linear programs, it seems natural that some of the algorithmic toolbox for linear programs can be applied to constraint problems. A key result along this line has been Thapper and Živný's [98] characterisation of bounded width VCSPs in terms of the Sherali-Adams hierarchy of linear programs. Namely, it states that for constraint languages

Γ that satisfy the bounded width condition (BWC), VCSP(Γ) is solved by its Sherali-Adams relaxation of level $(2, 3)$ (see Theorem 2.69).

It turns out that the connection between VCSPs and LPs is also quite useful from a descriptive complexity point of view. Since solutions to explicitly given linear programs are known to be definable in FPC (see Theorem 2.49), this gives us an avenue to obtain a definability result for the same class of bounded-width VCSPs.

In this section, we aim to provide some technical lemmas that establish the connection between VCSPs and linear programming in terms of logical definability. In particular, we show that by means of an FPC-interpretation, we can translate an instance of any constraint optmisation problem to the explicit representation of its 0–1 linear program (Section 2.4.1), as well as its Sherali-Adams relaxation of level $(2, 3)$ (Definition 2.67).

Formally, we show the following two lemmas.

Lemma 4.4. *Let Γ be a constraint language. There is an* FPC *interpretation Θ of τ_{LP} in τ_Γ, such that for every* VCSP(Γ) *instance $\mathbf{I}$, $\Theta(\mathbf{I})$ is the explicit representation of the 0–1 program associated with $\mathbf{I}$.*

Proof. Let $I = (V, C)$ be given as the τ_Γ structure $\mathbf{I}$ with universe $\mathrm{dom}(\mathbf{I}) = V \dot\cup C \dot\cup B$. Our goal is to define a τ_{LP}-structure $\mathbf{P}$ representing $\mathrm{BLP}(I)$ as a tuple by (A, b, t), with constraints $Ax \leq b$, and cost vector t. In order to refer to elements of the domain D in our interpretation, we fix a bijection $dom : D \to \{1, \ldots, |D|\}$ between D and an initial segment of the natural numbers. Furthermore we use m to denote the maximum arity of any function $f \in \Gamma$.

In order to define the constraint matrix A, and the vectors b and c, we first need to show that the index sets of A, b, and c are definable in FPC. That is, for a linear program with variables U and constraints E, the constraint matrix is indexed by $E \times U$, while $b \in \mathbb{Q}^E$ and $t \in \mathbb{Q}^U$. In our case, the set of variables U consist of two sorts, defined by

$$\begin{aligned} U &:= \lambda \dot\cup \mu \\ \lambda &:= \{\lambda_{c,x} \mid c = (s, f, q) \in C, x \in D^{\mathrm{ar}(f)}\} \\ \mu &:= \{\mu_{v,a} \mid v \in V, a \in D\}. \end{aligned}$$

These sets can be defined by the following formulas.

$$\delta_\lambda(c, \chi) := \bigvee_{f \in \Gamma} \left(\exists y \in V^{\mathrm{ar}(f)} : R_f(y, c) \wedge \bigwedge_{1 \leq i \leq ar(f)} \bigvee_{a \in D} \chi_i = dom(a) \right),$$

$$\delta\mu(v, \alpha) := v \in V \wedge \bigvee_{a \in D} \alpha = dom(a)$$

Note that for constraints c that use functions with an arity less than m, this creates some additional duplicate variables. We can either identify them by defining

our equivalence relation ε_λ accordingly, or, in this case we can ignore them, as they will simply not be used.

For the set of linear constraints, recall the definition of the linear program associated with I from Section 2.4.1. Observe that the constraints resulting from the equalities of the form (2.1) can be indexed by the set

$$J_1 = \{j_{c,a,i,\beta} \mid c = (s, f, q) \in C, a \in D, i \in [\mathrm{ar}(f)], \beta \in \{0, 1\}\}.$$

Here we have one equality for each constraint $c = (s, f, q)$, every $a \in D$, and every index $i \in [\mathrm{ar}(f)]$. As a single equality results in two inequalities in the LP standard form, we need an additional bit β to address each of them. This index set J_1 can be defined in FPC.

$$J_1(c, \alpha, \iota, \beta) := c \in C \wedge \bigvee_{f \in \Gamma} \exists y \in V^{ar(f)} : R_f(y, c) \\ \wedge \bigvee_{a \in D} \alpha = dom(a) \wedge \iota \in [\mathrm{ar}(f)] \wedge \beta \in \{0, 1\}.$$

The constraints resulting from (2.2) can be indexed by

$$J_2 = \{j_{v,\beta} \mid v \in V, \beta \in \{0, 1\}\}.$$

Again, each equality constraint is split into two inequalities to fit the LP normal form. Similarly, the equality constraints from (2.3) are indexed by

$$J_3 = \{j_{c,x,\beta} \mid c = (y, f, q) \in C, x \notin \mathrm{Feas}(f), \beta \in \{0, 1\}\}.$$

All the above sets are clearly FPC-definable, and their disjoint union is the set of constraints E.

$$E = J_1 \dot\cup J_2 \dot\cup J_3.$$

The universe $\mathrm{dom}(\mathbf{P})$ is then the multi-sorted set $U \dot\cup E \dot\cup B'$ with index sets U and E to index the columns (variables) and rows (constraints) of the constraint matrix A, and a (large enough) domain for bit positions B'.

Finally, we set the entries of the matrix $A \in \mathbb{Q}^{E \times U}$ and the two vectors $b \in \mathbb{Q}^E$ and $c \in \mathbb{Q}^U$ according to the definition of the LP given in Section 2.4.1. It is not difficult to see that the values of A can be extracted from a given variable-constraint pair by means of simple case distinction. Similar case distinctions work for the vector b, and the entry of the cost vector t at position $\lambda_{c,x}$ is simply the weight of the constraint c. Hence, we can conclude that all these can be suitably defined in the logic FPC.

□

Lemma 4.5. *Let Γ be a constraint language. There is an* FPC *interpretation Θ of τ_{LP} in τ_Γ, such that for every* $\mathrm{VCSP}(\Gamma)$ *instance $\mathbf{I}$, $\Theta(\mathbf{I})$ is the explicit representation of $\mathrm{SA}_{(2,3)}(\mathbf{I})$.*

Proof. Let $I = (V, C)$ be given as the τ_Γ structure $\mathbf{I}$ with universe $\mathrm{dom}(\mathbf{I}) = V \dot\cup C \dot\cup B$. Without loss of generality, we assume C contains some (possibly 0-weight) constraint for every tuple $x \in V^k$ for all $k \leq 3$.

The construction of the FPC-interpretation is very similar to the one in Lemma 4.4. Again, we describe how the constraint matrix and vector are indexed.

We again define a τ_{LP}-structure $\mathbf{P}$ representing $\mathrm{SA}_{(2,3)}(I)$ as a tuple (A, b, t), with constraints $Ax \leq b$, and cost vector t. We fix a bijection $dom : D \to \{1, \ldots, |D|\}$ between D and an initial segment of the natural numbers in order to refer to elements of the domain D by number terms. Furthermore we use m to denote the maximum arity of any function $f \in \Gamma$.

The set of variables of $\mathbf{P}$ is defined by

$$U := \{\lambda_{c,x} \mid c = (s, f, q) \in C, x \in D^{\mathrm{ar}(f)}\}.$$

As a FPC-formula, this can be defined as follows.

$$\delta_U(c, \mu_1, \ldots, \mu_m) = \bigvee_{f \in \Gamma} \left(\exists y \in V^{\mathrm{ar}(f)} : R_f(y, c) \wedge \bigwedge_{1 \leq i \leq m} \bigvee_{a \in D} \mu_i = dom(a) \right).$$

For the set of linear constraints, recall the definition of the Sherali-Adams relaxation of level $(2, 3)$ (Definition 2.67). The constraints resulting from the equalities of the form (2.6) can be indexed by the set

$$\begin{aligned} J_1 = \{ j_{c,d,a,\beta} \mid & c = (x, f, q), d = (y, g, r) \in C, \mathrm{ar}(g) \leq 3, a \in D^{\mathrm{ar}(g)}, \beta \in \{0, 1\} \\ & \exists S \subseteq [\mathrm{ar}(f)] : y = \pi_S(x) \}. \end{aligned}$$

Here we have one equality for each pair of constraints $c = (x, f, q)$ and $d = (y, g, r)$ and every $a \in D^{\mathrm{ar}(g)}$, where $\mathrm{ar}(g) \leq 3$, and y is a projection of x. This is defined in FPC as follows.

$$\begin{aligned} J_1(c, d, \alpha, \beta) = c, d \in C \wedge & \bigvee_{f \in \Gamma} \exists x \in V^{\mathrm{ar}(f)} : \wedge R_f(x, c) \\ & \wedge \bigvee_{\substack{g \in \Gamma; \\ \mathrm{ar}(g) \leq 3}} \bigvee_{S \subseteq [3]} \exists y \in V^{\mathrm{ar}(g)}, y = \pi_S(x) \wedge R_g(y, d) \\ & \bigwedge_{1 \leq i \leq m} \bigvee_{a \in D} \alpha_i = dom(a) \quad \wedge \quad \beta \in \{0, 1\}. \end{aligned}$$

Note that since the arity $\mathrm{ar}(g)$ is bounded by 3, we can write the existential quantifier over the projections $\pi_S, S \subseteq [\mathrm{ar}(g)]$ as a big disjunction over all subsets $S \subseteq [3]$.

The constraints resulting from (2.7) can be indexed by the set

$$J_2 = \{ j_{c,\beta} \mid c \in C, \beta \in \{0, 1\} \},$$

and constraints from (2.8) are indexed by

$$J_3 = \{ j_{c,x,\beta} \mid c = (y, f, q) \in C, x \notin \mathrm{Feas}(f), \beta \in \{0, 1\} \}.$$

We also have the non-negativity constraint for each variable, indexed simply by the variable set U. All sets are clearly FPC-definable, and again, we obtain as their disjoint union the set of constraints E.

$$E = J_1 \dot\cup J_2 \dot\cup J_3 \dot\cup U.$$

The universe $\mathrm{dom}(\mathbf{P})$ is the three-sorted set $U \dot\cup E \dot\cup B'$ with index sets U and E for the columns and rows of the constraint matrix A, and a domain for bit positions B'.

The values of the entries can be extracted by means of case distinction, according to the definition of $\mathrm{SA}_{(2,3)}(I)$. □

4.3 VCSPs definable in FPC

Having established the necessary technical machinery in the previous sections, we now turn to the positive part of Theorem 4.1. Namely, we show that for constraint languages Γ that satisfy the bounded width condition (BWC), $\mathrm{VCSP}(\Gamma)$ is definable in fixed-point logic with counting.

Theorem 4.6. *Let Γ be a constraint language such that $\mathrm{Pol}^+(\Gamma)$ satisfies the BWC. Then, $\mathrm{VCSP}(\Gamma)$ is definable in* FPC.

Proof. The main ingredients to this result are the expressibility of bounded width VCSPs as linear programs (Theorem 2.69), the definability result for linear programs in FPC (Theorem 2.49), and Lemma 4.5.

Putting all pieces together, let Γ be a constraint language that satisfies the bounded width condition, and let $\mathbf{I}$ be an instance of $\mathrm{VCSP}(\Gamma)$. Then, by Theorem 2.69, the optimal value to $\mathbf{I}$ is exactly the optimal value to $\mathrm{SA}_{(2,3)}(\mathbf{I})$. By Lemma 4.5, $\mathrm{SA}_{(2,3)}(\mathbf{I})$ is definable from $\mathbf{I}$, and by Theorem 2.49, the optimal value of any linear program is definable from its explicit representation, all in FPC. Composing these interpretations, we obtain the reduction as required. □

4.4 VCSPs not definable in $\mathcal{C}^\omega$

On the other side of the dichotomy we have that for those constraint languages Γ that are not of bounded width, $\mathrm{VCSP}(\Gamma)$ is not definable in $\mathcal{C}^\omega$. This result is obtained by using Lemma 4.3 to lift the undefinability result for classical CSPs (Theorem 2.66) to the framework of VCSPs.

Theorem 4.7. *Let Γ be a constraint language such that $\mathrm{Pol}^+(\Gamma)$ does not satisfy the BWC. Then, $\mathrm{VCSP}(\Gamma)$ is not definable in $\mathcal{C}^\omega$.*

Proof. This result follows from Theorem 2.66 once the reduction in Lemma 4.3 is established. For the sake of contradiction, assume Γ to be a constraint language

such that $\mathrm{Pol}^+(\Gamma)$ does not satisfy the BWC, but $\mathrm{VCSP}(\Gamma)$ *is* definable in $\mathcal{C}^\omega$. Now, pick Δ to be a relational constraint language with $\mathrm{Pol}(\Delta) = \mathrm{Pol}^+(\Gamma)$. Clearly, the constraint language Δ also does not satisfy the BWC. However, by Lemma 4.3, we have $\mathrm{CSP}(\Delta) \leq_{\mathrm{FPC}} \mathrm{VCSP}(\Gamma)$, and hence $\mathrm{CSP}(\Delta)$ is also definable in $\mathcal{C}^\omega$. This is a contradiction to the undefinability result of Theorem 2.66. □

Chapter 5

Constraint optimisation in the Lasserre hierarchy

This chapter contains material published in [40, 41]. The main results have been published in [41]

When faced with a difficult optimisation problem, it is often helpful to first try to solve the problem approximately instead of trying to obtain the exact optimum. In many cases, a good approximation is already good enough for the problem at hand; in others, approximations can give you a good starting point. Often, applying simple heuristics to the problem can already yield useful approximations. In the case of integer linear programs for instance, we can drop the integrality constraints to obtain a tractable rational linear program whose solution gives us an approximation to the original optimum. The gap between this approximate solution and the optimal solution to the integer program is commonly referred to as an *integrality gap*.

The concept of *relaxation hierarchies* is a generalisation of this approach. A relaxation hierarchy defines a systematic way to transform (or to *relax*) a given integer linear program into a series of rational integer or semidefinite programs whose solutions approximate the original optimum with increasing accuracy. To date, commonly used relaxation hierarchies include those of Lovasz-Shrijver [84], Sherali-Adams [94], and Lasserre [82]. Out of these, the Lasserre hierarchy is known to be the strongest. For an integer program of size n, the Lasserre hierarchy of level t defines a semidefinite program of size $n^{O(t)}$ whose feasible region, projected on the original variables, includes the solutions to the integer program. When $t = n$, this projection is in fact exactly the convex hull of the solutions to the integer program, and solving the Lasserre relaxation at this level always yields the exact solution, instead of a mere approximation. Still, in many cases, the integrality gap between the Lasserre relaxation and the original integer program vanishes at a level much smaller than n. In those cases, we obtain substantially faster algorithms for solving the original problem, simply by solving its Lasserre relaxation of a low level t.

In this chapter we establish a novel connection between constraint problems, logic, and the Lasserre hierarchy. First, we show that under certain technical as-

sumptions, we can define for any explicitly given semidefinite program its optimal value in the fixed-point logic with counting (FPC). This is a generalisation and extension of the work by Anderson, Dawar, and Holm [3, 4] that shows a similar result for linear programs (see Theorem 2.49). Formally, the definability result established here reads as follows.

Theorem 5.1. *Let instances of an SDP be given by (A, b, C) and an error parameter δ, with $A \in \mathbb{Q}^{M\times(V\times V)}$, $b \in \mathbb{Q}^M$, $C \in \mathbb{Q}^{V\times V}$, and $\delta > 0$. Its feasible region is denoted by $\mathcal{F}_{A,b}$. Let $\mathbf{I}$ be the relational representation of this SDP. Then, there is an* FPC*-interpretation Φ of τ_{mat} in $\tau_{\mathrm{SDP}} \dot\cup \tau_{\mathbb{Q}}$ such that $\Phi(\mathbf{I})$ defines a relational representation of $X \in \mathbb{Q}^{V\times V}$, such that*

- *if $\mathcal{F}_{A,b}$ is non-empty and bounded, then X is a δ-close and δ-maximal solution;*
- *otherwise X is unspecified.*

We apply this theorem to study the power of the Lasserre hierarchy applied to constraint optimisation problems. In particular, we show a linear lower bound on the minimum level t in the Lasserre hierarchy at which the integrality gap vanishes for a certain finite-valued constraint problems. The hardness criterion here is defined using a novel measure of complexity, called the *counting width*. Using the classification of finite-valued CSPs by Thapper and Živný (Theorem 2.70) we then show that VCSPs satisfying this criterion are in fact exactly those that can not be solved exactly by the basic linear programming relaxation. Together with the technical framework established in Chapter 4, this paints again a dichotomy.

Formally, this is expressed as follows. Recall the definitions of $\mathrm{BLP}(I)$ and $L_\Gamma(n)$ from Section 2.4.

Theorem 5.2. *Let Γ be a finite-valued constraint language. Then, either every instance I of* $\mathrm{VCSP}(\Gamma)$ *is solved exactly by* $\mathrm{BLP}(I)$*; or $L_\Gamma(n) \in \Omega(n)$.*

The study of relaxation hierarchies applied to constraint problems has some history. Grigoriev [54], as well as independently later, Schoenebeck [93] established linear lower bounds on the required Lasserre levels for a variety of constraint problems, including a number of MAXCSPs. The work here generalises some of these results into the larger framework of finite-valued CSPs. Very recently, Thapper and Živný also proved linear lower bounds for general VCSPs in the Sherali-Adams as well as the Lasserre hierachies [97], by formulating gadget reductions between $\mathrm{VCSP}(\Gamma)$ and $\mathrm{VCSP}(\Delta)$ for constraint languages Γ and Δ whose sets of polymorphisms share certain algebraic closure properties. We note that while the result in [97] generalises the lower bound result presented here, it uses significantly different techniques and does not examine the definability questions that arise from the viewpoint of descriptive complexity.

This chapter is structured as follows. We start in Section 5.1 by establishing that under certain technical assumptions, we can define the optimal value of any explicitly given semidefinite program within the logic FPC. This is shown by first formulating

a separation oracle for SDPs in FPC, and then by reducing the optimisation problem to the separation problem. The reduction is based on using the *ellipsoid method* as a black box, and extends some of the ideas in [3]. Then, in Section 5.2 we apply this definability result for SDPs to establish linear lower bounds for the number of Lasserre relaxations needed to solve finite-valued VCSP(Γ) exactly. For this we introduce a complexity measure called *counting width* and show a linear relation between the counting width of a constraint problem and the number of Lasserre levels needed to solve it exactly.

5.1 Definability of semidefinite programming

In this section we aim to show that semidefinite programs can be defined in FPC, subject to some technical assumptions. More precisely, we mean by "defined" that there is an FPC-interpretation that takes a relational representation of a (rational) SDP, and maps it to a relational representation of its optimal value. Due to the nature of semidefinite programs, there are some technical caveats to this statement here: First, since solutions to rational SDPs are potentially irrational, we are interested in solving the *weak optimisation problem* (see Definition 2.71). That is, as an input we accept an additional rational error parameter δ, and are seeking a δ-close and δ-optimal solution. Second, we require that the feasible region of the input SDP is *non-empty* and *bounded*. We note that the above assumptions are the same as for the polynomial time tractability of SDPs. That is, we show that whenever an SDP can be weakly optimised in polynomial time, then we can define it in the logic FPC. So far it seems unlikely that above assumptions can be relaxed further (without a breakthrough progress in computer science theory), as determining whether a given SDP has a non-empty feasible region in general is already an NP-hard problem in itself.

Our result is largely achieved by extending the methods of Anderson, Dawar, and Holm [3, 4] where the authors show an analogous result for the case of linear programming, as restated in Theorem 2.49. The central piece there is a formulation of the ellipsoid method for polyhedra in FPC. That is, the authors show that the reduction from the optimization problem to the separation problem for polyhedra can be accomplished in FPC. They then show that the separation problem for linear programs is itself definable in FPC. We follow the same two-step process and extend their techniques to prove Theorem 5.1.

5.1.1 Separation Oracle

A key part in proving Theorem 5.1 is to show that we can express a weak separation oracle for SDPs in FPC. That is, we aim to prove the following lemma.

Lemma 5.3. *There is an* FPC*-interpretation* Φ *of* τ_{mat} *in* $\tau_{\mathrm{SDP}} \dot\cup \tau_{\mathbb{Q}}$ *that does the following:*

An instance of the separation problem is given by (A, b, Y)*, and an error parameter* δ*, with* $A \in \mathbb{Q}^{M\times(V\times V)}$*,* $b \in \mathbb{Q}^M$*,* $Y \in \mathbb{Q}^{V\times V}$*, and* $\delta > 0$*. Let* $\mathbf{I}$ *be the relational representation of this instance.*

If $\mathcal{F}_{A,b}$ *is non-empty and bounded, then* $\Phi(\mathbf{I})$ *defines a relational representation of* $S \in \mathbb{Q}^{V\times V}$*, such that*

- *if* $S = 0$*, then* Y *is* δ*-close to* $\mathcal{F}_{A,b}$*;*
- *otherwise* $\langle S, Y\rangle + \delta > \max\{\langle S, X\rangle \mid X \in \mathcal{F}_{\mathcal{A},b}\}$.

Recall Algorithm 1 from Section 2.4, which describes a simple algorithm for the separation problem for semidefinite regions. While this algorithm is conceptually simple and efficient, it assumes that all computations can be calculated with infinite precision, even when numbers are irrational. Realistically, for the case of SDPs, we can only expect for an algorithm to solve the *weak* version of the separation problem, that is, to answer the query approximately within some error bound. Another issue of Algorithm 1 is that it depends on the order of constraints of the SDP. When a query violates several of the SDP constraints, the algorithm picks one of them to serve as a separation plane. This manner of choice is inherently not expressible in our logic.

In order to define a (weak) separation oracle in FPC, we still keep the basic idea behind Algorithm 1. However, we have to make two key modifications. First, we modify the algorithm to deal with finite precision arithmetic in the calculations. Second, we ensure that every step in the algorithm must be definable in FPC. We realise both points in Algorithm 2.

Algorithm 2 Weak separation oracle for semidefinite programs

Input: $\mathcal{A} = \{A_1, \ldots, A_m \in \mathbb{Q}^{V\times V}\}$, $b \in \mathbb{Q}^m$, $Y \in \mathbb{Q}^{V\times V}$, $\delta \in \mathbb{Q}$ such that $\delta > 0$.
Output: Solves weak separation problem on $\mathcal{F}_{\mathcal{A},b}$, Y, and δ.

function SEPARATION($\mathcal{A}$,b,Y,δ):
  $n \leftarrow |V|$
  $\mathcal{V} \leftarrow \{i \in [m] \mid \langle A_i, Y\rangle > b_i\}$
  if $\mathcal{V}$ is non-empty **then**
    $v \leftarrow \sum_{i\in\mathcal{V}} A_i$
    return $\frac{v}{\|v\|_\infty}$
  Approximate eigenvalues $\{\tilde{\lambda}_1, \ldots, \tilde{\lambda}_n\}$ of Y up to precision $\frac{\delta}{8n^2}$
  if there is $\tilde{\lambda}_i$ with $\tilde{\lambda}_i < \frac{\delta}{4n^2}$ **then**
    $K \leftarrow \{k \in [n] \mid \mathrm{LP}_k(Y, \tilde{\lambda}_i, \frac{\delta}{4n^2})$ has a solution.$\}$
    For $k \in K, v_k \leftarrow$ solution of $\mathrm{LP}_k(Y, \tilde{\lambda}_i, \frac{\delta}{4n^2})$
    $v \leftarrow -\sum_{k\in K} v_k v_k^T$
    return $\frac{v}{\|v\|_\infty}$
  return ACCEPT

As part of the algorithm we use $\mathrm{LP}_k(Y, \tilde{\lambda}, \epsilon)$ to denote the linear program:

$$\begin{aligned} &\min \sum_{i \in V} s_i && \text{subject to} \\ &(Y - \tilde{\lambda} I) v_i \leq s_i && \forall i \in V \\ &(Y - \tilde{\lambda} I) v_i \geq -s_i && \forall i \in V \\ &v_k = 1 \\ &-1 \leq v_i \leq 1, 0 \leq s_i \leq \epsilon && \forall i \in V. \end{aligned}$$

Compared to Algorithm 1, we find several key modifications in Algorithm 2. Apart from accepting an additional input δ that controls the precision of the output, we also changed the calculations of the separation vector in Line 5 and from Line 10 on to make these operations expressible in FPC.

To understand the purpose of these changes, let us first break down the linear programming step in the eigenvector calculation on Line 10. As in Algorithm 1, the goal is to find an eigenvector that corresponds to a negative eigenvalue of the input matrix Y. However, this presents a challenge. In general, the eigenvectors corresponding to some eigenvalue λ are not uniquely defined. Not only can we scale eigenvectors by an arbitrary scalar, but in the case of an eigenvalue of higher multiplicity, we have to choose a representative from a whole multidimensional eigenspace. To resolve this choice problem, we reformulate the problem as a linear program and rely on Theorem 2.49 to express this step in FPC.

More specifically, as approximate solutions are acceptable for Algorithm 2, it is enough if we can approximate these eigenvalues and eigenvectors. Hence, we can use a linear programming step to find a vector that approximates the eigenvector corresponding to a given (approximate) eigenvalue. That is, $\mathrm{LP}_k(Y, \tilde{\lambda}, \epsilon)$ aims to minimize the L^1-norm $\|(Y - \tilde{\lambda} I)v\|_1$, with the additional constraint that the k-th component of the solution vector v is fixed to 1. These additional constraints exist to ensure that the otherwise valid and optimal solution $s = 0$ and $v = 0$ is excluded. A solution to $\mathrm{LP}_k(Y, \tilde{\lambda}, \epsilon)$, if it exists, is then guaranteed to yield a non-zero vector v, such that $\|(Y - \tilde{\lambda} I)v\|_1 \leq n\epsilon$. In the other direction, if such a non-zero vector v exists, then it has at least one non-zero component v_i. Then, then there is some $\mu \in \mathbb{Q}$ such that μv is a solution to $\mathrm{LP}_k(Y, \tilde{\lambda}, \epsilon)$.

While any solution to any of $\mathrm{LP}_k(Y, \tilde{\lambda}, \epsilon)$ would work, it is not possible in FPC to choose a single solution to be used. To sidestep this issue of choice, we use a common trick: We take an average of all solutions. Namely, in Step 12 of Algorithm 2, we return

$$S := -\frac{\sum_{k \in K} v_k v_k^T}{\| \sum_{k \in K} v_k v_k^T \|_\infty},$$

where $K \subseteq [n]$ is the set of indices for which $\mathrm{LP}_k(Y, \tilde{\lambda}, \epsilon)$ has a solution, and v_k is an optimal solution for LP_k. As each $-v_k v_k^T$ is already a separating hyperplane for Y, by linearity, their sum is also a separating hyperplane. Note that the sum $\sum_{k \in K} v_k v_k^T$ is never the zero matrix, as the diagonal of each v_k is non-negative and

contains at least one non-zero entry. As we know by Theorem 2.49 that explicit linear programs are expressible in FPC, we can then conclude that these steps are expressible in FPC.

A similar averaging trick is performed in Line 5: In Algorithm 1, we have to choose a violated constraint from an unordered set of constraints, which is in general not possible to express in FPC. However, the explicit choice of a constraint can again be avoided by summing all violated constraints. Namely, if an input matrix Y violates a non-empty set of constraints with indices $\mathcal{V} := \{i \in [m] \mid \langle A_i, Y\rangle > b_i\}$, then by linearity, it also violates the constraint $\langle \sum_{i\in\mathcal{V}} A_i, Y\rangle \leq \sum_{i\in\mathcal{V}} b_i$. Note that our initial assumptions of Lemma 5.3 state that the feasible region is non-empty. As a consequence, it follows that the sum $\sum_{i\in\mathcal{V}} A_i$ is never the zero vector: If it were, then the constraint $\langle \sum_{i\in\mathcal{V}} A_i, Y\rangle = 0 \leq \sum_{i\in\mathcal{V}} b_i$ could not be satisfied by any point Y, contradicting the feasibility assumption. Hence, we can use $\sum_{i\in\mathcal{V}} A_i$ as the normal of a hyperplane separating Y. This sum avoid the choice problem is expressible in FPC.

Finally, we rely on some previous results to conclude that Algorithm 2 is definable as an FPC interpretation. For instance, it has been shown in [36, 65] that the basic vector and matrix operations, such as addition, multiplication, norm and even computing the characteristic polynomial can all be defined in FPC. Furthermore, in Line 7, we compute the eigenvalues of the input matrix Y up to a given precision $\frac{\delta}{8n^2}$. This is possible in FPC since the logic is powerful enough to define the coefficients of the characteristic polynomial of definable matrices.

Proposition 5.4. *There is an* FPC *interpretation of $\tau_{\mathbb{Q}}$ in $\tau_{mat} \dot\cup \tau_{\mathbb{Q}}$ that for a given a matrix $A \in \mathbb{Q}^{V\times V}$ and a value $\delta \in \mathbb{Q}$ (in their relational representation) defines the value of the smallest eigenvalue of A up to a precision of δ.*

Proof. Holm [65] establishes that there is an interpretation in FPC by which we can obtain from A the coefficients $\alpha_1, \ldots, \alpha_n$ of the characteristic polynomial $p(x) = \det(xI - A) = x^n - \alpha_1 x^{n-1} + \ldots + (-1)^n \alpha_n$. Note that the coefficients are linearly ordered (by the power of their corresponding monomial), and hence by the Immerman-Vardi theorem, any polynomial time computable property can be defined in FPC, such as computing the smallest eigenvalue up to a precision δ. □

The correctness of the algorithm follows from some basic calculations. Consider an input instance $(\mathcal{A}, b, Y, \delta)$. If the algorithm accepts this input, then Y violated none of the inequalities $\langle A_i, Y\rangle \leq b_i$, and all eigenvalues of Y are non-negative, and we can conclude that Y is in fact inside the feasible region $\mathcal{F}_{\mathcal{A},b}$.

If the algorithm does not accept, then either some inequality $\langle A_i, Y\rangle \leq b_i$ is violated, in which case the algorithm produces a correct separating hyperplane; or some approximated eigenvalue $\tilde\lambda$ is smaller than $\frac{\delta}{4n^2}$. In the latter case, the linear optimization step in Line 11 finds a matrix $v = -\sum_{k\in K} v_k v_k^T$. Note that v always exists, as shown in the following proposition.

Proposition 5.5. *In the context of Algorithm 2, there is always at least one value for k such that $\mathrm{LP}_k(Y, \tilde{\lambda}, \frac{\delta}{4n^2})$ has a solution.*

Proof. Let λ be the actual eigenvalue of Y, such that $\tilde{\lambda} = \lambda + \epsilon$ for some error ϵ with $|\epsilon| \leq \frac{\delta}{4n^2}$, and let v be an eigenvector for λ with $\|v\|_\infty = 1$. We then have

$$(Y - \tilde{\lambda}I)v = (Y - \lambda I - \epsilon I)v.$$

and hence for each component $i \in V$ we have

$$|((Y - \tilde{\lambda})v)_i| = |((Y - \lambda I - \epsilon I)v)_i| \leq |\epsilon| < \frac{\delta}{4n^2},$$

which shows that v is a solution of $\mathrm{LP}_k(Y, \tilde{\lambda}, \frac{\delta}{4n^2})$. □

Finally, we show that the matrix S returned on Line 12 is indeed a weak separation normal.

Proposition 5.6. *If Algorithm 2 returns on Line 12, then it holds*

$$\langle S, Y \rangle + \delta > \max_{X \in \mathcal{F}_{\mathcal{A},b}} \langle S, X \rangle.$$

Proof. Oberserve first that the right hand side of the inequality is always larger or equals to zero, as X is always positive semidefinite. Hence, it is sufficient for us to show $\langle S, Y \rangle > -\delta$. Let $v := -\sum_{k \in K} v_k v_k^T$, and let $e_k := Y v_k - \tilde{\lambda} v_k$. We have

$$\begin{aligned}
\langle S, Y \rangle = \frac{\langle v, Y \rangle}{\|v\|_\infty} &= -\frac{\langle \sum_{k \in K} v_k v_k^T, Y \rangle}{\|v\|_\infty} \\
&= -\sum_{k \in K} \frac{v_k^T Y v_k}{\|v\|_\infty} \\
&= -\sum_{k \in K} \frac{v_k^T (\tilde{\lambda} v_k + e_k)}{\|v\|_\infty} \\
&= -\sum_{k \in K} \tilde{\lambda} \frac{\|v_k\|^2}{\|v\|_\infty} + \frac{v_k^T e_k}{\|v\|_\infty}
\end{aligned}$$

Since each v_k is a solution of $\mathrm{LP}_k(Y, \tilde{\lambda}, \frac{\delta}{4n^2})$, we have $\|v_k\|^2 \leq n$, and $v_k^T e_k \leq \frac{\delta}{4n}$. Furthermore, we know $\|v\|_\infty \geq 1$, and $\tilde{\lambda} < \frac{\delta}{4n^2}$. Putting all this together, we have

$$-\sum_{k \in K} \tilde{\lambda} \frac{\|v_k\|^2}{\|v\|_\infty} + \frac{v_k^T e_k}{\|v\|_\infty} \geq -\sum_{k \in K} \frac{\delta}{4n^2} n + \frac{\delta}{4n} \geq -\sum_{k \in K} \frac{\delta}{2n} \geq -\frac{\delta}{2}.$$

□

This shows that we can define a weak separation oracle for SDPs in FPC. For the next step, we show that the reduction from weak optimization to separation, using the ellipsoid method, can be defined in FPC as well.

5.1.2 Reducing Optimization to Separation

In this section we construct an FPC-reduction from the weak optimization problem to the weak separation problem for SDPs, by formalizing the ellipsoid method in logic.

Lemma 5.7. *If there is a* FPC*-interpretation expressing the weak separation problem for the feasible region of a given SDP, then there is a* FPC*-interpretation which expresses the weak semidefinite optimization problem.*

The above result is known from [3] for the case of LPs instead of SDPs. The overall algorithm here follows a similar line as the one for linear programs, however with a couple of key extensions necessary to cope with semidefinite regions.

The main idea behind the construction is to repeatedly apply the separation oracle to define a linear order on the set of variables, and once a sufficient order is obtained, to apply the Immerman-Vardi theorem to define the ellipsoid method. This is achieved by defining a series of increasingly fine equivalence relations on the variable set V, specified by so-called *foldings* that we formalize below. Intuitively, these partitions are obtained in the following way: In the beginning, every element of V resides in the same equivalence class. However, there may be some inputs on which the separation oracle returns a vector d with different values d_u and d_v for $u, v \in V$, which distinguishes the two elements u and v, say $d_u < d_v$. In subsequent iterations, u and v are put in different equivalence classes, say $\mathcal{E}_u$ and $\mathcal{E}_v$, and we define an order on the classes to reflect the revealed order of d_u and d_v, i.e. $\mathcal{E}_u < \mathcal{E}_v$. This process is repeated until we obtain a sufficiently refined ordered partition of V. Once such a partition of V is obtained, we are able to apply the ellipsoid method on the set of equivalence classes. As the equivalence classes themselves are ordered, this is definable in FPC by the Immerman-Vardi theorem. Finally, it can be shown that any (weakly) optimal solution to a sufficiently folded SDP (where the variables are equivalence classes) is also (weakly) optimal for the orignal input.

We start by defining the notion of *folding*.

Definition 5.8. Let A be a non-empty set. For $k \leq |A|$, we call a surjective mapping $\sigma : A \to [k]$ an *index map*. Furthermore, for each $i \in [k]$ we define $A_i := \{a \in A \mid \sigma(a) = i\}$.

For a vector $x \in \mathbb{Q}^A$, the *almost-folded* vector $[x]^{\tilde{\sigma}} \in \mathbb{Q}^k$ is given by

$$([x]^{\tilde{\sigma}})_i := \sum_{a \in A_i} x_a, \text{ for } i \in [k].$$

Its *folded* vector $[x]^{\sigma} \in \mathbb{Q}^k$ is given by

$$([x]^{\sigma})_i := [x]^{\tilde{\sigma}}_i / |A_i|, \text{ for } i \in [k].$$

For a vector $\hat{x} \in \mathbb{Q}^k$, its *unfolded* vector $[\hat{x}]^{-\sigma} \in \mathbb{Q}^A$ is given by

$$([\hat{x}]^{-\sigma})_a := \hat{x}_i, \text{ with } a \in A_i, \text{ for all } a \in A.$$

For a given index map $\sigma : A \to [k]$ and a vector $x \in \mathbb{Q}^A$, we say x *agrees* with σ when for all $a, b \in A$ $\sigma(a) = \sigma(b)$ implies $x_a = x_b$. The notion also extends in a natural way to sets $S \subseteq \mathbb{Q}^A$, simply by defining the folded set $[S]^\sigma := \{[s]^\sigma \mid s \in S\}$. This can be seen as a projection of S into the (ordered) k-dimensional space $\mathbb{Q}^k$. In our case, where we reason about matrices $M \in \mathbb{Q}^{V \times V}$, index maps are simply applied over the set of pairs $A = V \times V$. That is, an index map is a mapping $\sigma : (V \times V) \to [k]$ where $k \leq |V|^2$.

There are some useful properties of the folding operation that allow us to infer some information about the geometry of a folded set from its original.

Proposition 5.9 ([3])**.** *Let $\sigma : A \to [k]$ be an index map, x, c vectors in $\mathbb{Q}^A$, where c agrees with σ. Then,*

$$\langle c, [[x]^\sigma]^{-\sigma} \rangle = \langle c, x \rangle = \langle [c]^{\tilde{\sigma}}, [x]^\sigma \rangle.$$

Proposition 5.10 ([3])**.** *Let $\mathcal{P} \subseteq \mathbb{Q}^A$ be a polytope (or a cone) in $\mathbb{Q}^A$ and let $\sigma : A \to [k]$ be an index map. The folded set $[\mathcal{P}]^\sigma$ is a polytope (or a cone) in $\mathbb{Q}^k$.*

Since the feasible region of an SDP is the intersection of a polytope with the positive semidefinite cone, we obtain as a corollary to Proposition 5.10 that the result of folding the feasible region of an SDP stays a convex region. Next we show that a weak separation oracle of the original set either serves as an oracle for the folded set, or produces some vector that does not agree with the index map of the folding.

Proposition 5.11. *Let $\mathcal{F} \subseteq \mathbb{Q}^A$ be a convex set, and let $\sigma : A \to [k]$ be an index map. Given a vector $x \in \mathbb{Q}^A$ that is δ-close to $\mathcal{F}$ for some $\delta \geq 0$, the folded vector $[x]^\sigma \in \mathbb{Q}^k$ is also δ-close to the folded set $[\mathcal{F}]^\sigma$.*

Proof. Let $x = f + d$, where $f \in \mathcal{F}$ is some point in the set $\mathcal{F}$, and d the difference vector with $\|d\| \leq \delta$. By the definition of folding, we then have $[x]^\sigma = [f]^\sigma + [d]^\sigma$, where $[f]^\sigma$ is now a point in the folded set $[\mathcal{F}]^\sigma$. We can bound the norm of $[d]^\sigma$ by

$$\|[d]^\sigma\| = \sqrt{\sum_{i \in [k]} \left(\frac{1}{|A_i|} \sum_{a \in A_i} d_a \right)^2} \leq \sqrt{\sum_{i \in [k]} (\max_{a \in A_i} d_a)^2} \leq \sqrt{\sum_{a \in A} d_a^2} = \|d\|.$$

Since $\|d\| \leq \delta$, we have $\|[d]^\sigma\| \leq \delta$. □

Proposition 5.12. *Let $\mathcal{F} \subseteq \mathbb{Q}^A$ be a convex set, and let $\sigma : A \to [k]$ be an index map. Given vectors $s, y \in \mathbb{Q}^A$ where s agrees with σ, and $\langle s, y \rangle + \delta > \max\{\langle s, x \rangle \mid x \in \mathcal{F}\}$, it holds that $\langle [s]^\sigma, [y]^\sigma \rangle + \delta > \max\{\langle [s]^\sigma, x \rangle \mid x \in [\mathcal{F}]^\sigma\}$.*

Proof. Let $x \in \mathcal{F}$ be a point in $\mathcal{F}$ such that $\langle s, x \rangle$ is maximal. It follows from Proposition 5.9 that $[x]^\sigma$ is also a maximal point in $[\mathcal{F}]^\sigma$ with respect to $\langle [s]^\sigma, [x]^\sigma \rangle$. We then have

$$\langle [s]^\sigma, [x]^\sigma \rangle - \langle [s]^\sigma, [y]^\sigma \rangle = \langle [s]^\sigma, [x - y]^\sigma \rangle \leq \langle s, x - y \rangle = \langle s, x \rangle - \langle s, y \rangle < \delta.$$

For the first equality we use the fact that $[x - y]^\sigma = [x]^\sigma - [y]^\sigma$, and for the first inequality we use Proposition 5.9 to get $\langle [s]^\sigma, [x]^\sigma \rangle \leq \langle [s]^{\tilde{\sigma}}, [x] \rangle = \langle s, x \rangle$. □

Assume now we are given a convex set $\mathcal{F} \subseteq \mathbb{Q}^{V \times V}$ by means of a corresponding weak separation oracle. We maintain an index map $\sigma : V \times V \to [k]$, which is initially a constant function. Our version of the Ellipsoid method then proceeds through the following steps:

1. Initialise $y \in Q^{V \times V}$ as the zero vector, initialise some enclosing ellipsoid in $Q^{[k]}$.

2. Query the separation oracle on (y, δ). If it accepts, we are done, and output y. Otherwise let s be the returned separation normal.

3. If s does not agree with σ, refine σ, and restart at Step 1.

4. Otherwise, by Proposition 5.12, $[s]^\sigma$ is a separation normal for $[y]^\sigma$ and $[\mathcal{F}]^\sigma$. We perform one iteration of the Ellipsoid method to obtain a new ellipsoid with center $c \in \mathbb{Q}^{[k]}$.

5. Let $y := [c]^{-\sigma}$, and go to Step 2.

Note that once σ is sufficiently refined, Proposition 5.11 ensures that whenever the oracle accepts some input (y, δ), then the folded vector $[y]^\sigma$ is also δ-close to the folded set $[\mathcal{F}]^\sigma$. This happens after at most $|V|^2$ many iterations.

For a more detailed discussion, we refer to the procedure in [3], only substituting their blackbox for a separation oracle by the one we obtained from Section 5.1.1, which also includes the definition of the refinement procedure. This then concludes the proof for Lemma 5.7.

Together with the result from Section 5.1.1 that the weak separation oracle for the feasible region of an explicitly given SDP can be defined in FPC, this now almost establishes our main result of Theorem 5.1. A small technicality still remains: We assume as a condition in Theorem 5.1 that the feasible region of the given SDP instance is bounded and non-empty, while the original reduction given in Theorem 2.37 assumes the region to be bounded and full-dimensional. However, by means of a simple preprocessing step, a non-empty region can be turned into a full-dimensional one.

Assume we are given an SDP with a feasible region of $\mathcal{F} = \{X \in \mathbb{Q}^{V \times V} \mid X \succeq 0, \langle A_i, X \rangle \leq b_i, A_i \in \mathcal{A}, b_i \in b\}$, and we want to find a δ-close and δ-maximal point of $\mathcal{F}_{\mathcal{A},b}$ with respect to some objective matrix C. We can then define an enlarged feasible region $\mathcal{F}' := \{X \in \mathbb{Q}^{V \times V} \mid X \succeq 0, \langle A_i, X \rangle \leq b_i + \frac{\epsilon}{\max(1, \|A_i\| \|C\|)} \sqrt{\frac{\mu}{2n}}\}$ for some $\epsilon > 0$, where we use μ as a measure of the sharpest angle at which two hyperplanes of $\mathcal{F}$ intersect,

$$\mu := \min_{|\langle A_i, A_j \rangle| \neq 1} \frac{1 - \langle A_i, A_j \rangle}{\|A_i\| \|A_j\|}.$$

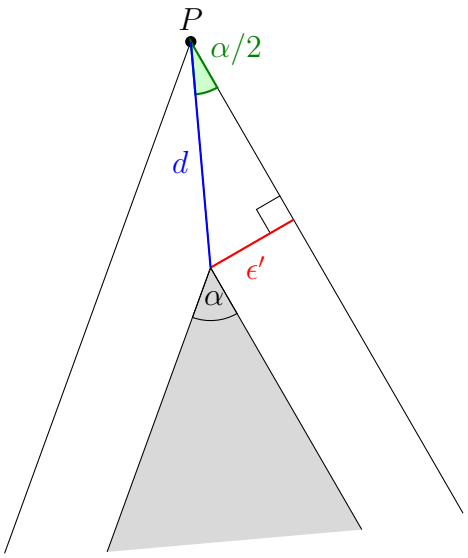
Proposition 5.13. *If $\mathcal{F}$ is non-empty, then $\mathcal{F}'$ is full-dimensional.*

Proof. Let $X_0 \in \mathcal{F}$, and let $\epsilon' := \frac{\epsilon}{\max(1,\|A_i\|\|C\|)}\sqrt{\frac{\mu}{2n}}$. By construction, $\mathcal{F}'$ contains the intersection of the (full-dimensional) ball $\mathcal{B}(X_0, \epsilon')$ with the semidefinite cone $\mathcal{S} := \{X \mid X \succeq 0\}$. Since $\mathcal{S}$ is convex and full-dimensional, and contains X_0, this means that there is a full-dimensional neighbourhood around X_0 in $\mathcal{S}$. This neighbourhood intersects a non-zero volume of $\mathcal{B}(X_0, \epsilon')$, and hence $\mathcal{F}'$ is full-dimensional. □

Proposition 5.14. *Any point in $\mathcal{F}'$ is at most ϵ far away from $\mathcal{F}$, i.e.*

$$\max_{X' \in \mathcal{F}'} \min_{X \in \mathcal{F}} \|X' - X\| \leq \epsilon.$$

Proof. Let again $\epsilon' := \frac{\epsilon}{\max(1,\|A_i\|\|C\|)}\sqrt{\frac{\mu}{2n}}$. Take a maximal point X' of $\mathcal{F}'$ such that $\langle A_i, X' \rangle = b_i + \epsilon'$. Then, either X' is above a face of $\mathcal{F}$ and the point $X' - \epsilon' \frac{A_i}{\|A_i\|}$ is in $\mathcal{F}$, in which case we have shown $\|X' - X\| \leq \epsilon$; or X' is in the neighbourhood of an intersection of two (or more) hyperplanes of $\mathcal{F}$. For the sake of simplicity, let us first assume that $\mathcal{F}$ is a two-dimensional polytope. The following figure illustrates the geometric situation in this case.



Above, the grey area is the inside of $\mathcal{F}$, while the outer lines represent the bounding hyperplanes of the enlarged region $\mathcal{F}'$. Let A_i and A_j be the normals of the two intersecting hyperplanes of $\mathcal{F}$, and assume that A_i and A_j form the sharpest intersection in $\mathcal{F}$. That is, $(A_i, A_j) = \operatorname{argmax}_{|\langle A_k, A_l \rangle| \neq 1} \frac{|\langle A_k, A_l \rangle|}{\|A_k\|\|A_l\|}$. We can see that the distance of the farthest point in $\mathcal{F}'$ to $\mathcal{F}$ depends on the angle α at which the bounding hyperplanes A_i and A_j intersect. More specifically, the distance d of the point P to $\mathcal{F}$ can be calculated by

$$\frac{\epsilon'}{d} = \sin\frac{\alpha}{2} = \sqrt{\frac{1-\cos\alpha}{2}} = \frac{1}{\sqrt{2}}\sqrt{\frac{1-\langle A_i, A_j \rangle}{\|A_i\|\|A_j\|}},$$

$$d = \epsilon' \sqrt{\frac{2\|A_i\|\|A_j\|}{1 - \langle A_i, A_j \rangle}}.$$

Generalising to higher dimensions, we can bound the distance of any point of $\mathcal{F}'$ to $\mathcal{F}$ by assuming a displacement of at most d in every dimension. That is, we can bound the distance of X' to $\mathcal{F}$ by

$$\min_{X \in \mathcal{F}} \|X' - X\| \leq \sqrt{n} \cdot \epsilon' \sqrt{\frac{2\|A_i\|\|A_j\|}{1 - \langle A_i, A_j \rangle}}.$$

Plugging in ϵ', we get $\min_{X \in \mathcal{F}} \|X' - X\| \leq \epsilon$.

□

We see that $\mathcal{F}'$ is a non-empty, full-dimensional convex set, where every point is at most ϵ far away from the original set $\mathcal{F}$. Furthermore, it holds that

$$\max_{X \in \mathcal{F}'} \langle C, X \rangle \leq \max_{X \in \mathcal{F}} \langle C, X \rangle + \epsilon.$$

Hence, any δ-close and δ-maximal point of $\mathcal{F}'$ is also a $\delta + \epsilon$-close and $\delta + \epsilon$-maximal point of $\mathcal{F}$. Consequently, by choosing ϵ sufficiently small, we can simply perform the optimization over the full-dimensional set, which is covered by Theorem 2.37.

This concludes the proof of Theorem 5.1. Note that the conditions on the feasible region of the definable SDP instances are readily satisfied for instance by those arising from finite-valued CSPs: The variables only range in $[0, 1]$, and there always exists a feasible solution. In fact, any (even non-optimal) assignment in the VCSP gives rise to a feasible solution of the 0–1 LP instance.

5.2 Lasserre lower bounds

In the previous section we have established that the optimal value of an explicitly given SDP can be defined within the logic FPC. Here, we apply this definability result to obtain a lower bound result in terms of a more algorithmic complexity measure. Namely, we aim to show that those finite-valued CSPs that are not solved exactly solution by their *basic linear program* (see Section 2.4.1) need in fact a linear number of levels of the Lasserre hierarchy to be captured.

In order to connect this lower bound result to the earlier definability result for SDPs we proceed in two significant steps. First, we show that the instances of VCSP(Γ) that are not solved by BLP(I) are also hard in a different complexity measure, called the *counting width* ν_Γ, which we introduce in the next section. This complexity measure is closely related to definability in finite-variable counting logic. In the second step we then establish that those VCSPs of large counting width also need a large number of levels in the Lasserre hierarchy to be captured. Hence, in the next two sections we aim to prove the following two lemmas, which together imply the result stated in Theorem 5.2.

Lemma 5.15. *Let Γ be a finite-valued constraint language. If there are instances I of* VCSP(Γ) *that are not solved by* BLP(I)*, then $\nu_\Gamma(n) \in \Omega(n)$.*

Lemma 5.16. *For any finite-valued constraint language Γ, $L_\Gamma \in \Omega(\nu_\Gamma)$.*

Finally, we note that the results presented here are restricted to *finite-valued* CSPs, as opposed to the more general VCSPs that allow constraints with infinite cost. Our insistence on the finite-valued variant has a technical reason: Finite-valued CSPs are always feasible, while general VCSPs are not, i.e. when every assignment has infinite cost. Our proof technique is specifically suited for the finite-valued variant as this guarantees that the resulting SDPs in the Lasserre hierarchy are non-empty. Using very different techniques, a similar lower bound result has been recently proven by Thapper and Živný for general VCSPs [97].

5.2.1 Counting width

Recall the logic $\mathcal{C}^k$, the infinitary logic with counting quantifiers consisting of those formulas that can be written using at most k distinct variables (see Section 2.5.3). Using different values for k, this gives us a family of logics that form a natural hierarchy, with fragments that allow a larger number of distinct variables being strictly more expressive than those that allow fewer. This motivates a certain complexity measure: For a fixed class of structures, we can determine the minimum number of distinct variables needed to define it in first-order logic with counting terms to measure its complexity. This notion is captured by the *counting width* of a class.

Definition 5.17. For any class of structures $\mathscr{C}$, the *counting width* of $\mathscr{C}$ is the function $\nu_{\mathscr{C}} : \mathbb{N} \to \mathbb{N}$ where $\nu_{\mathscr{C}}(n)$ is the minimum value k such that there is a formula ϕ in $\mathcal{C}^k$, such that for any structure $\mathbf{A}$ with $|\mathrm{dom}(\mathbf{A})| \leq n$, $\mathbf{A} \models \phi \Leftrightarrow \mathbf{A} \in \mathscr{C}$.

For the sake of convenience we write ν_Γ instead of $\nu_{\mathrm{VCSP}(\Gamma)}$. From the definition it is clear that $\nu_{\mathscr{C}} = \Omega(n)$ for any class $\mathscr{C}$. Moreover, it follows that $\nu_{\mathscr{C}}$ is bounded by a constant if and only if $\mathscr{C}$ is definable by a sentence in $\mathcal{C}^k$ for some fixed k, or in other words $\mathscr{C}$ is definable in $\mathcal{C}^\omega$. Since every class that is definable in FPC is also definable in $\mathcal{C}^\omega$, this means that if a class $\mathscr{C}$ is definable in FPC, then it must also have bounded counting width. Note that the converse is not true in general as there are even undecidable classes $\mathscr{C}$ that are definable in $\mathcal{C}^\omega$, and for which $\nu_{\mathscr{C}}$ is bounded by a constant. However, the results from Chapter 4 show that the converse does hold in the special case of general VCSPs. In fact, we have a dichotomy:

Proposition 5.18. *Let Γ be a (general) valued constraint language.* VCSP(Γ) *is either definable in* FPC *and ν_Γ is bounded by a constant; or ν_Γ in unbounded.*

Proof. Theorem 4.1 describes a definability dichotomy: Either VCSP(Γ) is definable in FPC; or it not definable in $\mathcal{C}^\omega$. The result follows then from the discussion above. □

The following proposition illustrates how the counting width of classes behaves together with FPC-reductions.

Proposition 5.19. *Let $\mathscr{C}_1$ and $\mathscr{C}_2$ be two classes of structures, such that $\mathscr{C}_1 \leq_{\mathrm{FPC}} \mathscr{C}_2$ by some* FPC*-reduction Θ. Furthermore, let $\theta : \mathbb{N} \to \mathbb{N}$ be defined as $\theta(n) = \max_{\mathbf{A} \in \mathscr{C}_1; |\mathbf{A}| \leq n} |\Theta(\mathbf{A})|$. Then $\nu_{\mathscr{C}_1}(n) \in O(\nu_{\mathscr{C}_2}(\theta(n)))$.*

Proof. Given any structure $\mathbf{A}$ (in the vocabulary of $\mathscr{C}_1$) of size n, the corresponding structure $\Theta(\mathbf{A})$ has size at most $\theta(n)$. Let $k := \nu_{\mathscr{C}_2}(\theta(n))$, then there is a formula ϕ in $\mathcal{C}^k$ for which $\Theta(\mathbf{A}) \models \phi \Leftrightarrow \Theta(\mathbf{A}) \in \mathscr{C}_2$. By composing ϕ with Θ, we obtain a formula ϕ' in $\mathcal{C}^{mk}$ that satisfies $\mathbf{A} \models \phi' \Leftrightarrow \mathbf{A} \in \mathscr{C}_1$, where m is the width of Θ. Hence, $\nu_{\mathscr{C}_1}(n) \leq m \cdot \nu_{\mathscr{C}_2}(\theta(n)) \in O(\nu_{\mathscr{C}_2}(\theta(n)))$. □

We continue by relating some known results in the CSP literature to counting width, and use them together to prove Lemma 5.15. Namely, we aim to show a linear lower bound for the counting width of those finite-valued CSPs that are not solved by the basic linear program. That is, if VCSP(Γ) is not solved by the BLP relaxation, we know that the MAXCUT problem reduces to it. Our argument proceeds by showing that (i) MAXCUT has linear counting width; and (ii) there is a linear size FPC-reduction from MAXCUT to VCSP(Γ), if it is not solved by its BLP relaxation. By Proposition 5.19 this suffices to prove our claim. Step (ii) relies on the dichotomy result of Thapper and Živný (Theorem 2.70).

For (i), we consider a chain of reductions that is also already present in the literature. We consider the problem 3-LIN: An instance of 3-LIN consists of a set of variables V, and two sets of equations, E_0 and E_1. Each equation in E_0 has the form $a \oplus b \oplus c = 0$, where $\oplus$ denotes addition modulo 2, and $a, b, c \in V$. Similarly, each equation in E_1 has the form $a \oplus b \oplus c = 1$. The problem is then to determine whether there is an assignment $h : V \to \{0, 1\}$ such that all equations are satisfied.

The following fact is known due to Atserias et al. [5].

Proposition 5.20. $\nu_{3-\mathrm{LIN}}(n) \in \Omega(n)$.

Proof. In [5] Atserias et al. show a lower bound of for the counting width of the problem 3-LIN that is proportional to the tree-width of the instance. More precisely, they show a construction that transforms any given graph $G = (V, E)$ with tree-width t into a pair of 3-LIN instances (I, I'), each having $O(|V|)$ variables, such that I is satisfiable, but I' is not, and no C^t formula distinguishes between them. It follows then by Theorem 2.53 that 3-LIN is not definable in $\mathcal{C}^t$.

The claim then follows by picking a class of graphs that have linear tree-width. Such graphs exist, for instance in the class of 3-regular expander graphs [2]. □

As a direct consequence, we obtain that 3-SAT also has linear counting width.

Proposition 5.21. $\nu_{3-\mathrm{SAT}}(n) \in \Omega(n)$.

Proof. Given an instance (V, E_0, E_1) of 3-LIN, we replace each equation $a \oplus b \oplus c = 0$ by the four clauses containing a, b, c that have an even number of negated literals, i.e. $(a \vee b \vee c)$, $(\neg a \vee \neg b \vee c)$, $(a \vee \neg b \vee \neg c)$, and $(\neg a \vee b \vee \neg c)$. Similarly, each equation $a \oplus b \oplus c = 1$ is replaced by the four clauses of a, b, c that have an odd number of negated literals. This results in a 3-SAT instance that is satisfiable if and only if the original 3-LIN instance was satisfiable. Clearly, this is a linear size reduction that can be implemented in FPC. □

To conclude that MAXCUT has linear counting width, we argue that there is a FPC-reduction from 3-SAT to MAXCUT that increases the size of instances at most linearly. The construction is essentially a translation of the classical polynomial time reduction into logic, which first reduces 3-SAT to 4-NAESAT (not-all-equals SAT), and then 4-NAESAT to MAXCUT. A k-ary not-all-equals clause $\mathrm{NAE}(l_1, \ldots, l_k)$ is true if and only if not all literals evaluate to the same value, i.e. $\neg(l_1 = l_2 = \ldots = l_k)$. The k-NAESAT problem is to decide whether a given conjunction of k-NAE clauses has a satisfying assignment.

Proposition 5.22. $\nu_{4-\mathrm{NAESAT}}(n) \in \Omega(n)$.

Proof. Given a 3-SAT instance I with variable set V, we consider a 4-NAESAT instance J over the variable set $V \dot\cup \{z\}$, i.e. the variables of J are exactly those of I, plus a fresh variable z. Observe that for any literals l_1, l_2, l_3, a not-all-equal clause $\mathrm{NAE}(l_1, l_2, l_3, 0)$ is true if and only if the 3-SAT clause $(l_1 \vee l_2 \vee l_3)$ is true. Hence, we define the clauses in J to contain $\mathrm{NAE}(l_1, l_2, l_3, z)$ for every clause $(l_1 \vee l_2 \vee l_3)$ in I.

The instance J is satisfiable if and only if I is satisfiable: Whenever there is a satisfying assignment for I, the same assignment extended with $z = 0$ is also a satisfying assignment for J. In the other direction, if there is a satisfying assignment for J, there is always a satisfying one that sets $z = 0$, since negating every variable does not change the value of a NAE-clause. This reduction is linear size, and is easily implemented in first-order logic. □

Proposition 5.23. $\nu_{3-\mathrm{NAESAT}}(n) \in \Omega(n)$.

Proof. Observe that we can split every 4-NAESAT clause $\mathrm{NAE}(a, b, c, d)$ into two 3-NAESAT clauses $\mathrm{NAE}(a, b, z)$ and $\mathrm{NAE}(\neg z, c, d)$ for some fresh variable z. This gives us a linear size reduction that can be implemented in first-order logic. □

An instance of weighted MAXCUT is a tuple (V, E, w, t) representing a graph (V, E), a weighting function $w : E \to \mathbb{Z}$ and a threshold value $t \in \mathbb{Z}$. The problem is then to decide whether the graph admits a weighted cut with a value of at least t. We encode MAXCUT instances as a relational structure $\mathbf{I}$ over the vocabulary $\tau_{\mathrm{MAXCUT}} = (E, W, T, <)$. The universe $\mathrm{dom}(\mathbf{I})$ is a two-sorted set $U = V \dot\cup B$, consisting of vertices V, and a set B of bit positions. In addition to the edge relation $E \subseteq V \times V$, there is a weight relation $W \subseteq V \times V \times B$ which encodes an

integer denoting the weight of an edge. Moreover, there is a unary relation $T \subseteq B$ encoding the threshold value. Finally, $<$ is the usual linear order on B.

For our purposes, we assume that instances of 3-NAESAT are encoded as relational structures over the vocabulary $\tau_{3-\text{NAESAT}} = (N_{000}, N_{001}, \ldots, N_{111})$ containing eight ternary relations. Each relation symbol corresponds to one type of clause, and the indices serve as flags that determine which literals are negated in a clause. For instance, a 3-NAESAT clause in the form $\text{NAE}(x, \neg y, z)$ would then be encoded as $(x, y, z) \in N_{010}$.

Proposition 5.24. $\nu_{\text{MAXCUT}}(n) \in \Omega(n)$.

Proof. The following construction transforms an instance $\mathbf{I} = (V, N_{000}, \ldots, N_{111})$ of 3-NAESAT into an equivalent MAXCUT instance $\mathbf{J} = (\text{dom}(J), E, W, T, <)$. Let m be the number of clauses in $\mathbf{I}$, and fix $M := 10m$. For each variable $v \in V$, we create two vertices in the graph, denoted v_0 and v_1, along with an edge (v_0, v_1) of weight M. For each tuple $(x, y, z) \in N_{ijk}$ we add a triangle between the vertices x_i, y_j, and z_k with edge-weight 1. Setting the cut threshold to $t := |V| \cdot M + 2m$ gives us an equivalent instance: If $\mathbf{I}$ is satisfiable, say by an assignment f, then the partition given by $p(v_i) = f(v) + i \bmod 2$ cuts through every edge of the form (v_0, v_1), and through two edges in every triangle, resulting in a cut value of $|V| \cdot M + 2m$. On the other hand, any cut of value larger or equal to $|V| \cdot M + 2m$ has to cut through all edges of the form (v_0, v_1), since it can only cut through two edges in each triangle. Hence, any such bipartition induces a satisfying assignment to the 3-NAESAT instance. The construction can be realised as the following FPC-interpretation.

The universe of $\mathbf{J}$ is defined as a two-sorted set $\text{dom}(\mathbf{J}) = U \dot\cup B$, consisting of vertices $U = V \times \{0, 1\}$ and bit positions $B = \{1, \ldots, \alpha\}$ for some sufficiently large α. In particular, α has to be chosen larger than $\log_2 t$. Since m is at most $|V|^3$, taking $\alpha = \log(|V|^4)$ suffices.

$$\delta_U(x) = x \in V \times \{0, 1\} \quad \text{and} \quad \delta_B(\mu) = \mu \in \{1, \ldots, \alpha\}$$

The edge relation is given by

$$\phi_E(x, y) = x_1 = y_1 \wedge x_2 \neq y_2 \\ \bigvee_{i,j,k \in \{0,1\}} \exists u, v, w \in V : N_{ijk}(u, v, w) \wedge x, y \in \{(u, i), (v, j), (w, k)\}.$$

Finally, the edge weights and the cut threshold are defined by

$$\phi_W(x, y, \beta) = x_1 = y_1 \wedge x_2 \neq y_2 \wedge \text{BIT}(1, \beta) \\ \vee \text{BIT}\left(10 \cdot \sum_{i,j,k \in \{0,1\}} \#_{u,v,w} N_{ijk}(u, v, w), \beta\right),$$

$$\phi_T(\beta) = \mathrm{BIT}\left((2 + 10 \cdot \#_v v \in V) \cdot \sum_{i,j,k \in \{0,1\}} \#_{u,v,w} N_{ijk}(u,v,w), \beta\right).$$

□

It now remains to show that there is also a linear size FPC-reduction from MAXCUT to those VCSP(Γ) that are not solved by their basic linear program. This relies on the following classification result of finite-valued constraint languages by Thapper and Živný [96], as well as the collection of FPC-definable reductions between VCSPs from Chapter 4.

We say that the property (XOR) holds for a finite-valued constraint language Γ over domain D if there are $a, b \in D, a \neq b$, such that $\langle\Gamma\rangle$ contains a binary function f with $f(a,b) = 1$ if $a = b$ and $f(a,b) = 0$ otherwise. For the definitions of $\langle\Gamma\rangle$ and Γ_c, revisit Definitions 2.19 and 2.59.

Lemma 5.25 ([96])**.** *Let Γ be a finite-valued constraint language.*

- *Either for each instance I of* VCSP(Γ)*, the optimal value of I is the same as* BLP(I)*;*
- *or property* (XOR) *holds for* Γ_c *(and* VCSP(Γ) *is* NP*-complete).*

Lemma 5.26. *Let Γ be a finite-valued constraint language. If* (XOR) *holds for* Γ_c*, then* MAXCUT $\leq_{\mathrm{FPC}}$ VCSP(Γ)*.*

Proof. Let $I = (V, E, w, t)$ be a given MAXCUT instance. We define an equivalent instance $J = (U, C, t')$ of VCSP($\langle\Gamma_c\rangle$) as follows. Since (XOR) holds for Γ_c, there are two distinct elements $a, b \in D$ for which $\langle\Gamma_c\rangle$ contains a binary function f, such that $f(a,b) = 1$ if $a = b$ and $f(a,b) = 0$ otherwise. By creating a variable for each vertex in V and adding a constraint $((u,v), f, w(e))$ for each edge $e = (u,v) \in E$, we obtain a VCSP with the same optimal solution. The threshold constant t' is then set to $t' = M - t$, where $M := \sum_{e \in E} w(e)$. This construction can be easily realised in FPC.

The above construction gives us an FPC-reduction from MAXCUT to VCSP($\langle\Gamma_c\rangle$). From here, we can use the results of the previous chapter to further reduce it to VCSP(Γ). Namely, Lemma 4.2 gives us VCSP($\langle\Gamma_c\rangle$) $\leq_{\mathrm{FPC}}$ VCSP(Γ), which proves our claim.

□

Combining the above results, we obtain a proof of Lemma 5.15: By Lemma 5.25, a finite-valued constraint language Γ either satisfies the property (XOR), or VCSP(Γ) is solvable by the basic linear program . If VCSP(Γ) is solvable by the BLP, then by Theorem 4.6 and 2.69, it is definable in FPC, and hence has bounded counting width. On the other hand, if Γ satisfies (XOR), then Lemma 5.26 shows that MAXCUT $\leq_{\mathrm{FPC}}$ VCSP(Γ), where the reduction is linear size. Since Proposition 5.24 states that $\nu_{\mathrm{MAXCUT}}(n) \in \Omega(n)$, it follows that in those cases also $\nu_\Gamma \in \Omega(n)$.

5.2.2 Lower bounds

In this section we aim to prove Lemma 5.16 and show a linear relationship between the counting width ν_Γ of VCSP(Γ) and the minimum number of levels in the Lasserre hierarchy required to solve VCSP(Γ) exactly. Together with Lemma 5.15 from the previous section, this concludes the proof of our dichotomy stated in Theorem 5.2.

The following proposition allows us to translate approximate solutions to exact ones. It quantifies the quality of approximation needed so that we can obtain the exact optimum of the original 0–1 problem by rounding an approximate optimum of its Lasserre SDP.

Proposition 5.27. *Let $I = (A, b, c)$ be a 0–1 linear program whose optimal value is integral. Its feasible region is given by $\mathcal{K} = \{x \in \mathbb{Q}^V \mid Ax \geq b\} \cap \{0,1\}^V$ with an objective vector $c \in \mathbb{Q}^V$. Furthermore let $\mathrm{Las}_t^\pi(\mathcal{K}) = \mathcal{K}^*$ for some t, and let $s \in \mathbb{Q}$ be the value of a $1/(4\max\{1, \|c\|\})$-close and $1/(4\max\{1, \|c\|\})$-maximal solution to $\mathrm{Las}_t(\mathcal{K})$ under the objective c. Then, by rounding s, we obtain the exact optimal value for I.*

Proof. Let s^* be the exact optimal value of $\mathrm{Las}_t(\mathcal{K})$, and by assumption, also the optimal value for I. We argue that $|s - s^*| \leq 1/4$, and hence rounding s yields s^*, since $s^* \in \mathbb{Z}$.

First, note that the condition that s is $1/(4\max\{1, \|c\|\})$-maximal means that $s + 1/(4\max\{1, \|c\|\}) \geq \max_{x \in \mathcal{K}} \langle c, x\rangle = s^*$. Hence, we have the lower bound $s \geq s^* - 1/4$.

The other direction follows from the fact that s is the value of a close solution, say, $s = \langle c, y\rangle$ for some $y \in \mathbb{Q}^V$. Since y is $1/(4\max\{1, \|c\|\})$-close to $\mathcal{K}^*$, it can be decomposed into $y = x + e$ where $x \in \mathcal{K}^*$ and $\|e\| \leq 1/(4\max\{1, \|c\|\})$. The value of y is then bounded by $s = \langle c, y\rangle \leq \max_{x \in \mathcal{K}} \langle c, x\rangle + \langle c, e\rangle \leq s^* + 1/4$. □

Next, we show that given an instance of VCSP(Γ), it is possible to define its corresponding Lasserre relaxation of any level in FPC, using only a linear number of distinct variables. To do this, we first prove that from an explicitly given 0–1 linear program, we can define its t-th level Lasserre relaxation in FPC, using linearly many variables. Lemma 4.4 then shows that there is also a suitable FPC-interpretation that defines the corresponding 0–1 linear program given an instance of VCSP(Γ). Composing these two interpretations then gives us an interpretation from VCSP(Γ) instances to explicit SDPs of Lasserre hierarchy.

Lemma 5.28. *There is an* FPC*-interpretation from τ_{LP} to τ_{SDP} that for an explicitly given 0–1 linear program with a feasible region of $\mathcal{K} \cap \{0,1\}$ expresses the explicit representation of $\mathrm{Las}_t(\mathcal{K})$, using at most $O(t)$ many variables.*

Proof. Let an instance I of a 0–1 program be given by a matrix $A \in \mathbb{Q}^{U \times V}$, and vectors $b \in \mathbb{Q}^U, c \in \mathbb{Q}^V$. In order to show that we can define the t-th level Lasserre relaxation from I, it suffices to show that we can define the matrices $M_t(y)$ and $S_t^u(y)$ from I for any $y \in \mathbb{Q}^{\wp_{2t+1}(V)}, u \in U$. In particular, we represent $M_t(y)$ as

a sequence of matrices $(\hat{M}_{t,q})_{q\in Q}$ where $Q := \wp_{2t+1}(V) \cup \{0\}$, such that $M_t(y) = \hat{M}_{t,0} + \sum_{q\in Q\setminus\{0\}} y_q \hat{M}_{t,q}$, and show that these matrices are definable. We represent $S^u_t(y)$ in an analogous way as $S^u_t(y) = \hat{S}^u_{t,0} + \sum_{q\in Q\setminus\{0\}} y_q \hat{S}^u_{t,q}$. Hence, here it suffices to argue that the matrices $(\hat{M}_{t,q})_{q\in Q}$ and $(\hat{S}^u_{t,q})_{q\in Q}$ are definable from I within FPC.

Observe that for the t-th level Lasserre relaxation the matrices $M_t(y)$ and $S^u_t(y)$ are indexed by the power set $\wp_t(V)$, and the feasible region itself lies in a vector space indexed by $\wp_{2t+1}(V)$. As we need these index sets in our interpretation, we first describe how to define the power sets $\wp_k(V)$ for some fixed k. Namely, we encode a set $S \in \wp_k(V)$ by a k-ary tuple $T \in (V \cup \{0\})^k$ that contains each of the elements in S once, and where the symbol 0 fills the rest of the positions. As there are up to $k!$ many tuples encoding the same set, we additionally define an equivalence relation $\tilde{=}_k$ on k-tuples that identifies two tuples if they are just permutations of each other. This can be defined by the following first order formulas.

$$\begin{aligned}\delta_{\wp_k}(x_1,\ldots,x_k) \;:=\; & \bigwedge_{i\in[k]} (x_i = 0 \vee x_i \in V) \\ & \bigwedge_{i,j\in[k]} (x_i = 0 \vee x_i \neq x_j),\end{aligned}$$

$$\tilde{=}_k(x_1,\ldots,x_k,y_1,\ldots,y_k) := \bigvee_{\pi\in Sym(k)} \bigwedge_{i\in[k]} x_i = y_{\pi(i)},$$

where $\delta_{\wp_k}$ defines the set of tuples encoding some element in $\wp_k(V)$, and $\tilde{=}_k$ defines a binary equivalence relation between those tuples. We use $Sym(k)$ to denote the set of permutations on $[k]$. From these definitions it is not hard to define basic set operations on the elements of $\wp_k(V)$. For instance, we can define a $4k$-ary relation $union_k$ that encodes the union of two sets $S, T \in \wp_k(V)$.

$$\begin{aligned}union_k(x,s,t) = & \bigwedge_{i\in[2k]} \left(x_i = 0 \bigvee_{j\in[k]} x_i = s_j \vee x_i = t_j \right) \\ & \bigwedge_{i\in[k]} \bigvee_{j\in[2k]} s_i = x_j \bigwedge_{i\in[k]} \bigvee_{j\in[2k]} t_i = x_j,\end{aligned}$$

where $x \in \delta_{\wp_{2k}}$, and $s, t \in \delta_{\wp_k}$. Since $union_k(x,s,t)$ simply encodes $(x = s \cup t)$, we continue using the latter more familiar notation for set operations. One point to note here is that all formulas so far are all defined using $O(k)$ many variables.

Now we can turn to the definition of the matrices $(\hat{M}_{t,q})_{q\in Q}$ and $(\hat{S}^u_{t,q})_{q\in Q}$. Each matrix $\hat{M}_{t,q}$ and $\hat{S}^u_{t,q}$ is indexed over the set $\delta_{\wp_t} \times \delta_{\wp_t}$. For $x, y \in \delta_{\wp_t}$ and $q \in \delta_{\wp_{2t+1}}$, their entries are given by

$$\hat{M}_t(q,x,y) = \begin{cases} 1 & \text{if } x \cup y = q \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\hat{S}_t^u(q,x,y) = \begin{cases} A_{u,v} & \text{if } \exists v \in V : x \cup y \cup \{v\} = q \\ -b_u & \text{if } x \cup y = q \\ 0 & \text{otherwise.} \end{cases}$$

By the above expressions $(\hat{M}_{t,q})_{q\in Q}$ and $(\hat{S}^u_{t,q})_{q\in Q}$ are definable in FPC using only $O(t)$ variables. From this, we obtain the full SDP in inequality standard form by merging the constraints $M_t(y) \succeq 0$ and $S_t^u(y) \succeq 0$ into one single constraint of the form $Z \succeq 0$.

□

Finally, we are now ready to prove Lemma 5.16.

Proof of Lemma 5.16. Let Γ be a fixed finite-valued constraint language. Recall that we write L_Γ for the minimum number of levels in the Lasserre relaxation needed to capture every instance of VCSP(Γ), ν_Γ for the counting width of VCSP(Γ), and τ_Γ for the relational signature of instances of VCSP(Γ).

The proof idea is as follows. The argument is by contradiction. Suppose that $L_\Gamma(n) \in o(\nu_\Gamma(n))$. Then, by composing the interpretations from Lemmas 4.4 and 5.28 and Theorem 5.1 we define a formula ϕ that decides membership for the decision version of VCSP(Γ) for instances of size n using only $o(\nu_\Gamma(n))$ many variables. However, this violates the assumed counting width bound of $\nu(n)$.

To be more precise, let Θ_t be the composition of the interpretations from Lemmas 4.4 and 5.28. That is, Θ_t is an interpretation of τ_{SDP} in τ_Γ that defines for a given VCSP instance $I = (V, C, w)$ the SDP of the t-th level of the Lasserre relaxation of the corresponding 0–1 linear program. Note that Θ_t is of width $O(t)$.

Note that the 0–1 linear programs corresponding to VCSP instances are always feasible, bounded in the 0–1 hypercube, and their optimum is always integral.

Suppose now $L_\Gamma(n) \in o(\nu_\Gamma(n))$, i.e. every instance $I = (V, C, w)$ of VCSP(Γ) could be captured by some Lasserre relaxation of level $t \in o(\nu_\Gamma(|V|))$. Hence, $\Theta_t(I)$ defines a Lasserre relaxation whose optimal value is exactly the optimal value to I.

Then, by Theorem 5.1 there is an interpretation Σ of τ_{vec} in $\tau_{\text{SDP}} \dot\cup \tau_{\mathbb{Q}}$ that defines δ-close and δ-maximal solutions to $\Theta_t(I)$. Using Proposition 5.27, setting δ as $\delta = 1/4|C|$ allows us to obtain the exact optimal value for $\Theta_t(I)$ (and equivalently, for I) by means of rounding. Both defining the value for δ as well as the rounding can be done in FPC. Hence composing Θ_t and Σ, we obtain an interpretation Φ of width $O(t)$ that defines for a given instance of VCSP(Γ) its optimal value.

Finally, using Φ it is not difficult to construct a FPC-formula ϕ using at most $O(t)$ many variables that decides membership for the decision version of VCSP(D, Γ): For an instance (I, t) we simply compare $\Phi(I)$ to t. Since we assumed $t \in o(\nu_\Gamma(|V|))$, ϕ also uses only $o(\nu_\Gamma(|V|))$ many variables. This is a contradiction to the definition of ν_Γ. □

Chapter 6

Pebble games and homomorphisms

This chapter contains material published in [1].

Within the theory of constraint satisfaction problems, or homomorphisms, the class of *bounded width* CSPs (see Section 2.6.2) is one of the most studied subclasses. There are several characterisations of this problem class, each coming from a different perspective: From an algorithmic view, bounded width CSPs are solvable by *local cosistency* methods; from an algebraic view, their polymorphisms satisfy the *bounded width condition*, and from a logical view, their complements are definable in the logic Datalog. Here, we focus on the logical point of view, and show a deeper connection between the theory of homomorphisms and one of the key notions of finite model theory, namely *pebble games*.

Pebble games are one of the key tools in studying the expressive power of logics. In particular, we focus on the so-called *existential k-pebble game* [75], which characterises the expressive power of existential positive k-variable infinitary logic ($\exists^+\mathcal{L}^k$, see Section 2.5.3), of which k-variable Datalog is a fragment. An instance of a pebble game is played on two structures A and B, between two players, typically called Spoiler and Duplicator. In the existential k-pebble game, there are k pairs of pebbles, with Spoiler and Duplicator each owning one pebble of a pair. In each round, Spoiler first places one of his pebbles on an element of A, and Duplicator responds by placing the corresponding pebble onto an element of B. The game continues if the current positions of the pebbles induces a *partial homomorphism* between A and B. That is, the mapping that maps the element under the i-th pebble of Spoiler to the element under the i-th pebble of Duplicator is a homomorphism on the substructures induced by the pebbled elements. The game ends and Spoiler wins if Duplicator can not respond in any way that maintains the partial homomorphism. On the other hand, if there is a way for Duplicator to keep the game going forever, then we say Duplicator wins. We call the plan that Duplicator uses to keep playing forever her *winning strategy*. This notion will be more formally developed later. We often write $A \to_k B$ if there exists a winning strategy for Duplicator in the existential k-pebble game on A and B, and $A \to_{\text{hom}} B$ if there is a homomorphism from A to B.

The correspondence between the existential k-pebble game and existential positive k-variable logic is as follows.

Theorem 6.1 ([74, 75])**.** *The following are equivalent:*

- *Duplicator has a winning strategy in the existential k-pebble game played on A and B.*
- *Every existential positive first-order sentence of k variables satisfied by A is also satisfied by B.*

In practice, this means that given a class $\mathcal{C}$, if there are structures A and B with $A \in \mathcal{C}$ and $B \notin \mathcal{C}$, and Duplicator has a winning strategy in the existential k-pebble game on A and B, then the class $\mathcal{C}$ can not be definable in $\exists^+\mathcal{L}^k$. This is a common technique to prove undefinability of a class. Consequently, if we do know that $\mathcal{C}$ is definable, and Duplicator wins on A and B, then $A \in \mathcal{C}$ implies $B \in \mathcal{C}$.

In this chapter, we formalise the notion of a Duplicator winning strategy in the existential k-pebble game, and treat them as our main objects of study. The main idea is as follows. Given a structure A, we define a new structure $\mathbb{T}_k A$, called the *k-unravelling* of A, that represents all potential move sequences of Spoiler playing in A. We can then define a Duplicator strategy for the game on A and B by giving a mapping from $\mathbb{T}_k A$ to B. We show that if and only if this mapping is a homomorphism, then the strategy is a winning strategy for Duplicator. We introduce this formalism in detail in Section 6.1, and prove basic properties.

In Section 6.2, we show that this formalism is also an elegant way to characterise another pebble game central in finite model theory: The *k-pebble bijection game*. This pebble game corresponds to the logic C^k, the k-variable logic with counting quantifiers. We show that by restricting winning strategies for the existential game to be bijective in a certain sense, we immediately obtain a characterisation of winning strategies for the bijection game. This establishes a nice link when viewing the existence of a winning strategy in those two games as *local approximations* of homomorphism and isomorphism respectively. As an application we construct, based on [5], a more general version of the Cai-Fürer-Immerman construction [26] using this framework.

Finally, we discuss potential future directions and open questions in Section 6.3.

The results and notions presented here have been further developed by Abramsky et al. [1], showing that the notions of k-unravellings and strategies have a very natural representation in category theory.

6.1 Unravellings, positions, and strategies

For the remainder of this chapter, when talking about structures, we implicitly mean relational structures over a finite signature σ. Recall that we write L^k for the k-variable fragment of first-order logic, $\exists^+\mathcal{L}^k$ for its existential positive fragment (no

negations or universal quantifiers), and C^k for the logic obtained by extending L^k with counting quantifiers. For two sequences s and t, we write $s \leq_{\mathrm{pre}} t$ to denote that s is a prefix of t. That is, there is a unique (possibly empty) sequence s', such that $ss' = t$. In this case, we call s' the suffix of s in t.

We begin by introducing some key notions. First, consider an instance of the existential k-pebble game over two structures A and B. Note that in this game, Spoiler is only allowed to place pebbles on the structure A. Hence, the set of possible move sequences for Spoiler actually only depends on A, and remains the same for any choice of target structure B. We formalise this set of possible Spoiler move sequences over a given structure A into the notion of the *k-unravelling* of A, written as $\mathbb{T}_k A$.

A single move of Spoiler on A can be represented as a pair (p, a), where $p \in [k]$ denotes which of the k pebbles Spoiler moves, and $a \in A$ tells on which element of A the pebble is placed. The set of all finite move sequences in A is then the set $([k] \times A)^+$, where any one sequence is in the form $s = [(p_1, a_1), \ldots, (p_n, a_n)]$. This set forms the universe of $\mathbb{T}_k A$.

The relations of $\mathbb{T}_k A$ are best illustrated if we first assume that A contains only a single, binary relation E^A. Then, $\mathbb{T}_k A$ also contains a single binary relation, $E^{\mathbb{T}_k A}$. For two sequences $s = [(p_1, x_1), \ldots, (p_m, x_m)]$ and $t = [(q_1, y_1), \ldots, (q_n, y_n)]$ it holds $E^{\mathbb{T}_k A}(s, t)$ if:

- s is a prefix of t, i.e. $s \leq_{\mathrm{pre}} t$;
- The pebble number p_m, does not appear in any move in the suffix of s in t;
- The elements in the last move of s and t are in E^A, i.e. $E^A(x_m, y_n)$.

Intuitively, from a sequence $s = [(p_1, a_1), \ldots, (p_m, a_m)]$, you have an edge to any sequence obtained by appending any number of moves to s, as long as you do not touch the pebble p_m, and make the final move onto an element next to a_m.

We generalise the above conditions for higher arity relations as follows. Given an m-ary relation R^A, we have $R^{\mathbb{T}_k A}(s_1, \ldots, s_m)$ if $s_1 \leq_{\mathrm{pre}} s_2 \leq_{\mathrm{pre}} \ldots \leq_{\mathrm{pre}} s_m$; for $i \in [m-1]$, the pebble number in the last move of s_i does not appear in any move in the suffix of s_i in s_m; and it holds $R^A(a_1, \ldots, a_m)$ where a_i is the element in the last move of s_i. If the signature of A contains multiple relations, each relation R^A induces a relation $R^{\mathbb{T}_k A}$ in $\mathbb{T}_k A$ by the above rules.

Example 6.2. Let A be a 3-element cycle, i.e. it is the structure with the universe $\{a, b, c\}$ over a single binary relation $E^A = (a, b), (b, c), (c, a)$. Then, the universe of $\mathbb{T}_2 A$ is the infinite set $[\{1, 2\} \times \{a, b, c\}]^+$. The relation $E^{\mathbb{T}_2 A}$ is also infinite, and for instance, contains the infinite path $[(1, a)]$, $[(1, a), (2, b)]$, $[(1, a), (2, b), (1, c)]$, $[(1, a), (2, b), (1, c), (2, a)]$,... Note that for instance there is no edge between the sequence $[(1, a), (2, b)]$ and $[(1, a), (2, b), (2, c)]$. This is prevented by the second condition in the definition.

The k-unravelling $\mathbb{T}_k A$ is defined over a single structure A, and represents all move sequences that a player can possibly play on it. Now we turn to the case where we have a fixed instance of the existential k-pebble game, and both A and B are given. We define the notion of a *position* in the game.

Definition 6.3. Given two structures A and B, a *position* in the existential k-pebble game on A and B is a partial function

$$\gamma : [k] \to (A \times B).$$

The domain of γ is the set of pebbles from $[k]$ that have been placed so far, and $\gamma(p) = (a, b)$ means that the pebble p is on the element $a \in A$ and on $b \in B$ in the position γ.

We define the *update operation* $\gamma[p \mapsto (a, b)]$ to yield a new position

$$\gamma[p \mapsto (a,b)](q) = \begin{cases} (a,b), & \text{for } p = q, \\ \gamma(q), & \text{otherwise.} \end{cases}$$

A position γ simply encodes the current state of a game, and describes which pebbles lie on which elements of A and B. If after one round of the game, Spoiler placed a pebble p on element a in A, and Duplicator responded with placing her pebble on b in B, then the new position is reflected by the update $\gamma[p \mapsto (a, b)]$. Note that the initial position is the empty partial function, and the domain of the current position grows whenever a previously unplaced pebble is used.

To describe how the state of one game changes over time, we make use of the transition notation $\gamma \xrightarrow{(p,a):b} \gamma'$ to denote $\gamma' = \gamma[p \mapsto (a, b)]$. Similarly, we use $\gamma \to \gamma'$ if $\gamma \xrightarrow{(p,a):b} \gamma'$ for some choice of p, a, and b.

Given the notion of positions, we are now ready to define what a (Duplicator) *strategy* in the existential k-pebble game is. We will introduce two different ways to define this concept: The first one relates strategies to a set of positions, and the second one defines strategies as mappings from the k-unravelling of A to B.

In the first view, a strategy is a set of positions that Duplicator is willing to visit throughout the course of the game. We call this view *strategies in positional form*.

Definition 6.4. Let $\Gamma_k(A, B)$ be the set of all positions in the existential k-pebble game on A and B. A *strategy in positional form* is a set $S \subseteq \Gamma_k(A, B)$ satisfying the following:

(i) The empty function is in S, i.e. $\emptyset \in S$;

(ii) For all $\gamma \in S$, $p \in [k]$, and $a \in A$, there is some $\gamma' \in S$ and $b \in B$, such that $\gamma \xrightarrow{(p,a):b} \gamma'$;

(iii) For all $\gamma \in S$, there is a finite sequence of positions $[\gamma_1, \ldots, \gamma_n]$, such that $\gamma_1 = \emptyset$, $\gamma_n = \gamma$, and $\gamma_i \to \gamma_{i+1}$ for all $i \in [n]$.

Note that in the above definition we have no condition that Duplicator must maintain a partial homomorphism between A and B after each round. We say instead that a position γ is *winning* (for Duplicator) if the relation $R_\gamma = \{(a, b) = \gamma(p) \mid p \in \mathrm{dom}(\gamma)\}$ is a partial homomorphism from A to B. A strategy S is then called a *winning strategy* if every position $\gamma \in S$ is winning.

The second way we define strategies is to view any mapping $f : \mathbb{T}_k A \to B$ as a strategy. The idea is that given a sequence $s \in \mathbb{T}_k A$ of Spoiler moves in A, Duplicator responds with placing the most recent pebble on $f(s)$ in B.

Definition 6.5. A *strategy in function form* is simply a mapping $f : \mathbb{T}_k A \to B$.

Let $s \in \mathbb{T}_k A$ with $s = [(p_1, a_1), \ldots, (p_n, a_n)]$, and let $s_i = [(p_1, a_1), \ldots, (p_i, a_i)]$ for $i \in [n]$. We define a mapping $\theta_f : \mathbb{T}_k \to \Gamma_k(A, B)$ as

$$\theta_f(s) = \gamma,$$

where γ is obtained by applying the sequence of n updates to the empty position $\emptyset \xrightarrow{(p_1,a_1):f(s_1)} \ldots \xrightarrow{(p_n,a_n):f(s)} \gamma$. We call θ_f the *position function* of the strategy f.

Given the position function θ_f it is straightforward to convert a strategy in function form to one in positional form. That is, for a given strategy f, we have its representation in positional form as

$$S_f = \{\theta_f(s) \mid s \in \mathbb{T}_k A\} \cup \{\emptyset\}.$$

The following lemma establishes this relationship in both directions.

Lemma 6.6. *Given a strategy $f : \mathbb{T}_k A \to B$ in function form, the set of positions S_f is a strategy in positional form. Conversely, for any positional strategy S, there is a mapping $f : \mathbb{T}_k A \to B$, such that $S_f = S$.*

Proof. For the first direction we simply check that S_f satisfies conditions $(i) - (iii)$ of Definition 6.4. By definition of S_f, point (i) is satisfied trivially. For (ii), take any $\gamma \in S_f$, and suppose that $\gamma = \theta_f(s)$ for some move sequence s. Then, any choice of a pebble p and element $a \in A$ results in an extended move sequence $s' = s[(p, a)]$, and a Duplicator response $f(s')$. We choose $\gamma' = \theta_f(s')$ and have $\gamma \xrightarrow{(p,a):f(s')} \gamma'$ as required. For (iii), we can show the reachability of every $\gamma = \theta_f(s))$ by induction on the length of s, since $\theta_f(s) \to \theta_f(s[(p, a)])$ for every pebble p and element a.

For the converse direction, we aim to define $f(s)$ from a given set S by induction on the length of s. A technical issue here is that a positional strategy S may be *non-deterministic*. That is, it is possible for a fixed position $\gamma \in S$, and fixed $p \in [k]$ and $a \in A$ to have two (or more) different $b_1, b_2 \in B$ such that both $\gamma \xrightarrow{(p,a):b_1} \gamma_1 \in S$ and $\gamma \xrightarrow{(p,a):b_2} \gamma_2 \in S$. However, the strategy $f : \mathbb{T}_k A \to B$ is a *determinisitic* mapping. We resolve this apparent contradiction by making use of the fact that different move sequences may end up in the same position.

In order to deal with the non-determinism, we fix a linear order $\leq_B$ on the elements of B. We define $f(s)$ inductively over the length of the sequence s. As the base case of the induction, consider all move sequences $s = [(p, a)]$ of length one. We then define $f(s)$ as $\min_{\leq_B}\{b \in B \mid \exists \gamma \in S : \emptyset \xrightarrow{(p,a):b} \gamma\}$. For the induction case, consider a move sequence $s[(p, a)]$. By induction, $f(s)$ is already defined, with $\theta_f(s) = \gamma \in S$. We define two sets

$$X := \{b \in B \mid \exists \gamma' \in S : \gamma \xrightarrow{(p,a):b} \gamma'\},$$

$$Y := \{b \in X \mid \exists t[(p, a)] \leq_{\text{pre}} s : \theta_f(t) = \gamma \wedge f(t[(p, a)]) = b\}.$$

Then, we define $f(s[(p, a)])$ as

$$f(s[(p, a)]) = \begin{cases} \min_{\leq_B}(X \backslash Y), \text{ if } X \backslash Y \neq \emptyset \\ \min_{\leq_B}(X), \text{ otherwise.} \end{cases}$$

By definition of X, $\theta_f(s[(p, a)]) \in S$ and hence $S_f \subseteq S$. It is left to check that $S \subseteq S_f$. Given $\gamma \in S$ we prove $\gamma \in S_f$ by induction over the length of the shortest transition sequence from $\emptyset$ to γ. The base case of a single move is easily checked from the earlier base case argument. Now, consider $\gamma' \in S$ with $\gamma \xrightarrow{(p,a):b} \gamma'$, and by induction $\gamma = \theta_f(s) \in S_f$, where s is taken as a minimal element of $\theta_f^{-1}(\gamma)$ (w.r.t. $\leq_{\text{pre}}$). Let X be the set defined as before, and assume that there are i elements in X strictly smaller than B (w.r.t. $\leq_B$). Now, consider the move sequence $s' = s[(p, a), \ldots, (p, a)]$ where the move (p, a) is appended $i+1$ times. By definition, $f(s') = b$, and thus $\theta_f(s') = \gamma'$. Hence, $\gamma' \in S_f$. □

We see that the positional and function strategies can be transformed into each other. Consistent with this transformation, we call a strategy in function form f *winning*, if its positional form S_f is winning.

With these notions defined we can now turn to the main result of this section: We show that there exists a winning strategy for Duplicator in the existential k-pebble game on A and B if and only if there is a homomorphism from $\mathbb{T}_k A$ to B.

In order to state this result precisely, it is convenient to extend our relational signature σ by a single binary relation I. A σ-structure A is then extended to a $\sigma \cup \{I\}$-structure by interpreting I as the identity relation, i.e. $I^A = \{(a, a) \mid a \in A\}$. We call $\sigma \cup \{I\}$-structures *structures with identity*. Note that the interpretation of I in the k-unravelling of A is not necessarily the identity. Instead, $I^{\mathbb{T}_k A}$ relates two move sequences s and t if their last move placed a pebble on the same element. This means that a homomorphism $f : \mathbb{T}_k A \to B$ must map s and t to the same element $b \in B$.

Theorem 6.7. *Let A and B be structure with identity. A strategy $f : \mathbb{T}_k A \to B$ is a winning strategy if and only if f is a homomorphism.*

Proof. We show that f is a homomorphism from $\mathbb{T}_k A$ to B if and only if for any position $\gamma \in S_f$, the relation $R_\gamma = \{(a, b) = \gamma(p) \mid p \in \mathrm{dom}(\gamma)\}$ is a partial homomorphism from A to B.

For the first direction, assume f is in fact a homomorphism. We show that any $\gamma = \theta_f(s)$ is winning by induction on the length of the sequence s. The base case of s consisting of a single move is easily checked. For the induction case, consider a sequence $s = [(p_1, a_1), \ldots, (p_n, a_n)]$, and let $s_i = [(p_1, a_1), \ldots, (p_i, a_i)]$ for $i \in [n]$. By induction, we know that all positions $\gamma_i = \theta_f(s_i)$ are winning. Assume now that the element pebbled in the last move, a_n, is incident to some set of pebbled elements $N \subseteq \{a_1, \ldots, a_n\}$ w.r.t. some relation R^A. That is, for each $a_j \in N$, there is some tuple in R^A containing both a_n and a_j. Now, $\mathbb{T}_k A$ is defined exactly such that for each $a_j \in N$, s_j is incident to s w.r.t. $R^{\mathbb{T}_k A}$ if and only if the pebble placed on a_n is different than the one used on a_j, i.e. $p_j \neq p_n$. Hence, since f is a homomorphism, $f(s)$ must be incident to all $f(s_j)$ for all $a_j \in N$, $p_j \neq p_n$. Since the choice of R^A was arbitrary, this holds for any relation in the vocabulary of A. This is exactly the condition needed to maintain that $\gamma = \theta_f(s)$ is a partial homomorphism.

Similarly, in the other direction, assume that f is not a homomorphism, and there are some sequences s and t, and some relation R, such that s and t are incident w.r.t. $R^{\mathbb{T}_k A}$, but $f(s)$ and $f(t)$ are not w.r.t. R^B. Let (p_s, a_s) and (p_t, a_t) be the last moves of s and t respectively. Now, since s and t are incident, by definition of $\mathbb{T}_k A$, so are a_s and a_t w.r.t. R^A. Furthermore, p_s does not occur in the suffix of s in t. This means that in $\gamma = \theta_f(t)$, a_s and a_t are pebbled elements that are incident w.r.t. R^A, while $f(s)$ and $f(t)$ are pebbled, but not incident w.r.t. R^B. Hence, γ can not be a winning position.

□

6.2 Game equivalences

In this section we apply the notions of k-unravellings and strategies to the k-pebble bijection game [63], the pebble game corresponding to the logic $\mathcal{C}^k$. To motivate our results, we first describe the view that the existential pebble game and the bijection game can serve as approximation for the homomorphism and isomorphism problems repsectively.

In a sense, the relation $A \to_k B$ as a coarsening, or *local approximation* of the homomorphism relation $A \to_{\mathrm{hom}} B$. If A is homomorphic to B by some homomorphism h, then certainly $A \to_k B$: The strategy $f = \epsilon \circ h$, where $\epsilon([(p_1, a_1), \ldots, (p_n, a_n)]) = a_n$, is a homomorphism from $\mathbb{T}_k A$ to B, and hence winning. The converse direction is however not true: There are structures A, B such that $A \to_k B$, but not $A \to_{\mathrm{hom}} B$. Still, the converse does hold in certain special cases, which, remarkably, are deeply connected to constraint satisfaction problems. Namely, it is known that if B has *width* k in the sense of Definition 2.62, then $A \to_k B$ in fact implies $A \to_{\mathrm{hom}}$. This connection between the local consistency algorithm for CSPs and the existential k-pebble game has been established by Kolaitis and Vardi [75].

Proposition 6.8 ([75]). *$A \to_k B$ if and only if the local consistency algorithm accepts on input A, B.*

Hence, the local consistency algorithm of width k, as described in Section 2.6.2, is a polynomial time algorithm for deciding the relation $\to_k$. This can serve as a kind of approximative homomorphism test, in the sense that $A \to_{\hom} B$ only if the test succeeds on A and B (although the test could succeed even if $A \not\to_{\hom} B$). As discussed in Section 2.6.2, this approximation becomes exact in the case of bounded width structures.

A similar situation exists when talking about the isomorphism relation $\simeq_{\text{iso}}$ instead of homomorphism. We can consider the equivalence relation under sentences of C^k: We say $A \equiv^{\mathrm{C}^k} B$ if every sentence in C^k is satisfied in A if and only if it is satisfied in B. We can view the relation $\equiv^{\mathrm{C}^k}$ as a local approximation of $\simeq_{\text{iso}}$: If $A \simeq_{\text{iso}} B$, then certainly $A \equiv^{\mathrm{C}^k} B$ for every k. Again, the converse is not true in general. However, there is again an algorithmic counterpart. The relation $A \equiv^{\mathrm{C}^k} B$ can be characterised in terms of whether a popular isomorphism test, the k-dimensional Weisfeiler-Leman algorithm, accepts on A and B [101]. The relation $\equiv^{\mathrm{C}^k}$ is also related to the notion of counting width introduced in Section 5.2.1. In fact, the counting width of a class $\mathscr{C}$ is just the value $\nu_{\mathscr{C}}(n)$ such that the class of stuctures in $\mathscr{C}$ of size n or less is closed under the equivalence relation $\equiv^{\mathrm{C}^k}$ for $k := \nu_{\mathscr{C}}(n)$.

In the previous section, we established that the existence of a winning strategy for the existential k-pebble game on A and B can be rephrased as the existence of a homomorphism from $\mathbb{T}_k A$ to B. The main result of this section is to show that the existence of a winning strategy in the k-pebble bijection game can be rephrased in terms of the existence of a *bijective* winning strategy in the existential pebble game. This notion is made more precise later. Equivalently, this condition also expresses that $\mathbb{T}_k A$ is *isomorphic* to $\mathbb{T}_k B$.

We start by recalling the k-pebble bijection game on two structures A and B. As the existential k-pebble game, an instance of the k-pebble bijection game is played on two structures A and B, by two players, Spoiler and Duplicator, each in the possession of k pebbles. Given a game position γ, a round proceeds as follows. First, Spoiler picks a pebble p. Duplicator then responds by choosing a bijection $h : A \to B$ such that for all pebbles $q \neq p$, if $\gamma(q) = (a, b)$, then $h(a) = b$. Spoiler then selects some pair of elements (a, b) with $h(a) = b$ and places the pebble p on it. The game continues if the position after the round γ' induces a partial isomorphism, i.e. the relation $R_{\gamma'} = \{(a, b) = \gamma'(p) \mid p \in \mathrm{dom}(\gamma')\}$ is a partial isomorphism. Again, the game ends and Spoiler wins if the game stops at some point, and Duplicator wins if she can keep the game running forever.

As mentioned earlier, the bijection game characterises equivalence in the k-variable logic with counting quantifiers C^k.

Theorem 6.9 ([63]). *Let A and B be two finite structures. Then the following are equivalent:*

- *Duplicator has a winning strategy in the k-pebble bijection game played on A and B.*
- $A \equiv^{\mathrm{C}^k} B$.

It turns out that winning strategies for the bijection game can be described in terms of strategies for the existential pebble game. For this, we introduce the notions of *injective*, *surjective*, and *bijective* strategies.

Definition 6.10. Let $f : \mathbb{T}_k A \to B$ be a strategy for the existential k-pebble game on A and B. For each move sequence $s \in \mathbb{T}_k \cup \{[]\}$ and pebble $p \in [k]$, we define a function $\psi_{f,s,p} : A \to B$ to yield

$$\psi_{f,s,p}(a) = f(s[(p, a)]).$$

We say f is an *injective* (or *surjective*) strategy, if $\psi_{f,s,p}$ is injective (or surjective) for all s and p. We say f is a *bijective* strategy if $\psi_{f,s,p}$ is bijective for all s and p, and for all positions $\gamma = \theta_f(s)$, the relation R_γ is a partial isomorphism.

We use the notation $A \to^i_k B$, $A \to^s_k B$, and $A \to^b_k B$ to mean that there is an injective, surjective, or bijective winning strategy in the existential k-pebble game respectively.

Observe that the relation $A \to^b_k B$ describes exactly the case where Duplicator can win in the k-pebble bijection game.

Lemma 6.11. *For two structures A and B, Duplicator has a winning strategy in the k-pebble bijection game if and only if $A \to^b_k B$.*

Proof. This follows by definition. Let $f : \mathbb{T}_k A \to B$ be a bijective winning strategy for the existential k-pebble game. In any round starting in position $\gamma \in S_f$, for any pebble p chosen by Spoiler, Duplicator can respond with the bijection $\psi_{f,s,p}$. Then, for any pair (a, b) with $b = \psi_{f,s,p}(a)$, the new resulting position is also in S_f. Since the empty position is in S_f by definition, Duplicator can keep the game running forever. Finally, since f is a bijective strategy, any position in S_f induces a partial isomorphism. □

In the usual terms of homomorphisms, it is a well known fact that for two finite structures A and B, if there are injective homomorphism from A to B and from B to A, then A and B are isomorphic. The same holds for surjective homomorphisms. The main result of this section shows that a similar situation also holds for the relation $\to_k$. Together with Lemma 6.11, this gives us an elegant characterisation of winning strategies in the k-pebble bijection game.

Theorem 6.12. *The following are equivalent:*

(i) $A \to^i_k B \wedge B \to^i_k A$.

(ii) $A \to^s_k B \wedge B \to^s_k A$.

(iii) $\mathbb{T}_k A \simeq_{\text{iso}} \mathbb{T}_k B$.

(iv) $A \to_k^b B$.

(v) *Duplicator has a winning strategy in the k-pebble bijection game played on A and B.*

Proof. For $(i) \Leftrightarrow (ii)$ note that A and B are assumed to be finite structures, and hence if there are injective mappings in both directions, then A and B must have the same number of elements. Hence, every injective map is in fact also surjective, and the equivalence holds by definition.

For $(i) \Rightarrow (iii)$, let $f : \mathbb{T}_k A \to B$ and $g : \mathbb{T}_k B \to A$ be two injective winning strategies witnessing $A \to_k^i B$ and $B \to_k^i A$. We define two mappings $f^* : \mathbb{T}_k A \to \mathbb{T}_k B$ and $g^* : \mathbb{T}_k B \to \mathbb{T}_k A$ as

$$f^*([(p_1, a_1), \ldots, (p_n, a_n)]) = [(p_1, f(s_1)), \ldots, (p_n, f(s_n))],$$

$$g^*([(p_1, a_1), \ldots, (p_n, a_n)]) = [(p_1, g(s_1)), \ldots, (p_n, g(s_n))],$$

with s_i denoting the initial sequence of length i, $[(p_1, a_1), \ldots, (p_i, a_i)]$. By Theorem 6.7, both f and g are homomorphisms, and by definition of the k-unravelling, so are f^* and g^*. Note that f^* and g^* are also injective, since $\psi_{f,s,p}$ and $\psi_{g,s,p}$ are injective for every choice of s and p. Furthermore, let $(\mathbb{T}_k A)_m$ and $(\mathbb{T}_k B)_m$ denote the substructures of $\mathbb{T}_k A$ and $\mathbb{T}_k B$ respectively restricted to move sequences of length up to m. It follows that for each m, the finite substructures $(\mathbb{T}_k A)_m$ and $(\mathbb{T}_k B)_m$ are isomorphic, since f^* and g^* restricted to these substructures are injective homomorphisms in each direction. By induction on m, it follows that $\mathbb{T}_k A$ and $\mathbb{T}_k B$ are also isomorphic.

For $(iii) \Rightarrow (iv)$, let f^* be an isomorphism $f^* : \mathbb{T}_k A \to \mathbb{T}_k B$. Consider now the strategies $f : \mathbb{T}_k A \to B$ and $g : \mathbb{T}_k B \to \mathbb{T}_k A$ defined as $f(s) = \epsilon(f^*(s))$, and $g(s) = \epsilon(f^{*-1}(s))$ with $\epsilon([(p_1, a_1), \ldots, (p_n, a_n)]) = a_n$. Note that both f and g are winning strategies, as they are homomorphisms. It follows that $\psi_{f,s,p}$ is a bijection for every sequence $s \in \mathbb{T}_k A$ and pebble p. Furthermore, any position $\gamma = \theta_s(f)$ is also in S_g, since $\theta_s(f) = \theta_{f^*(s)}(g)$. Moreover, $R_\gamma = \{(a, b) = \gamma(p) \mid p \in \mathrm{dom}(\gamma\}$ is a partial isomorphism on the pebbled elements, since R_γ and $R_\gamma^{-1} = \{(b, a) \mid (a, b) \in R_\gamma\}$ are injective homomorphisms in each direction. Hence, the strategy f witnesses $A \to_k^b B$.

For $(iv) \Rightarrow (i)$, let $f : \mathbb{T}_k A \to B$ be a bijective winning strategy witnessing $A \to_k^b B$. Then, f is also an injective winning strategy by definition. Similarly, since $\psi_{f,s,p}$ is a bijection, the strategy $g : \mathbb{T}_k B \to A$ defined by $g(s[(p, b)]) = \psi_{f,s,p}^{-1}(b)$ is also an injective winning strategy.

Finally $(iv) \Leftrightarrow (v)$ is the statement of Lemma 6.11. □

6.2.1 Cai-Fürer-Immerman construction

The original Cai-Fürer-Immerman construction (CFI), published in [26], is the key component to the classical result that there exists a polynomial-time decidable prop-

erty of graphs, that is not definable in fixed-point logic with counting. Later, the work of Atserias et al. in [5] generalised the original construction to show that the class of satisfiable systems of linear equations (which is decidable in polynomial time by Gaussian elimination) is not definable in the even stronger logic $\mathcal{C}^\omega$.

In this section, we use the notions and results of the previous sections to show that the original CFI-construction is part of a more general pattern to obtain counter-examples to definability in $\mathcal{C}^\omega$.

We will work in the relational signature of systems of three-variable linear equations over the $\mathbb{Z}_2$. That is, the signature consists of two ternary relations, R_0 and R_1. A structure A with relations R_0^A and R_1^A then represents a system of equations with the universe of A as variables, and that contains the equation $x \oplus y \oplus z = 0$ for each triple $(x, y, z) \in R_0$, and $x \oplus y \oplus z = 1$ for each $(x, y, z) \in R_1$. Furthermore, we fix a specific structure $Z2$, whose universe is $\{0, 1\}$, and whose relations are defined as $R_0^{Z2} = \{(x, y, z) \mid x \oplus y \oplus z = 0\}$ and $R_1^{Z2} = \{(x, y, z) \mid x \oplus y \oplus z = 1\}$. This is chosen such that $A \to_{\hom} Z2$ is equivalent to A being a satisfiable system of equations.

Lemma 6.13. *Let A be such that $A \to_k Z2$ for some fixed k. There are structures A_0 and A_1, such that $A_0 \equiv^{\mathrm{C}^k} A_1$, $A \to_{\hom} A_0$, and $A_1 \to_{\hom} Z2$.*

Proof. We define A_0 and A_1 as the following structures. Both have the same universe of $A \times \{0, 1\}$. In A_0, the relations are interpreted as

$$R_0^{A_0} = \{((a, i), (b, j), (c, l)) \mid (a, b, c) \in R_0^A \wedge i \oplus j \oplus l = 0\},$$

$$R_1^{A_0} = \{((a, i), (b, j), (c, l)) \mid (a, b, c) \in R_1^A \wedge i \oplus j \oplus l = 0\}.$$

In A_1, the relations are given by

$$R_0^{A_1} = \{((a, i), (b, j), (c, l)) \mid (a, b, c) \in R_0^A \wedge i \oplus j \oplus l = 0\},$$

$$R_0^{A_1} = \{((a, i), (b, j), (c, l)) \mid (a, b, c) \in R_1^A \wedge i \oplus j \oplus l = 1\}.$$

It is easily checked that the mapping $h(a) = (a, 0)$ is a homomorphism $A \to A_0$, and the mapping $h'(a, i) = i$ is a homomorphism $A_1 \to Z2$.

Now, let $f : \mathbb{T}_k \to Z2$ be a winning strategy witnessing $A \to_k Z2$. We aim to construct a bijective winning strategy $g : \mathbb{T}_k A_0 \to A_1$. This then proves $A_0 \equiv^{\mathrm{C}^k} A_1$ by Theorem 6.9 and Lemma 6.11. We define a bijection $\psi_{g,s,p} : A_0 \to A_1$ for every move sequence $s \in \mathbb{T}_k A_0 \cup \{[]\}$ and every pebble $p \in [k]$, given by

$$\psi_{g,s,p}(a, i) = (a, f(s^*[(p, a)]) \oplus i),$$

where

$$s^* = [(p_1, (a_1, i_1)), \ldots, (p_n, (a_n, i_n))]^* = [(p_1, a_1), \ldots, (p_n, a_n)].$$

Our strategy g is then defined as $g(s[(p, (a, i))]) = \psi_{g,s,p}(a, i)$.

Clearly, $\psi_{g,s,p}$ is a bijection. To see that every position $\gamma \in S_g$ induces a partial isomorphism, we check the relations in the pebbled substructure. For any position $\gamma = \theta_g(s)$, let γ^* be the position in the pebble game on A and $Z2$ given by

$\gamma^* = \theta_f(s^*)$. Furthermore, let $h_{\gamma^*} : A \times [k] \to Z2$ be defined as $h_{\gamma^*}(a,p) = b$ for $\gamma^*(p) = (a,b)$. Observe that due to the definition of the strategy g, we have $\gamma(p) = ((a,i),(a,h_{\gamma^*}(a,p) \oplus i))$ for every $p \in [k]$ and $(a,i) \in A_0$.

Assume now we are in a position γ and there are pebbles p_1, p_2, and p_3 on the elements (a,i), (b,j), and (c,l) in A_0, and $((a,i),(b,j),(c,l)) \in R_0^{A_0}$. It follows that $h_{\gamma^*}(a) \oplus h_{\gamma^*}(b) \oplus h_{\gamma^*}(c) = 0$ and $i \oplus j \oplus l = 0$. Hence, the triple $((a, h_{\gamma^*}(a,p_1) \oplus i),(b,h_{\gamma^*}(b,p_2) \oplus j),(c,h_{\gamma^*}(c,p_3) \oplus l))$ is in $R_0^{A_1}$ as required. For the case that $((a,i),(b,j),(c,l)) \in R_1^{A_0}$, it holds $h_{\gamma^*}(a) \oplus h_{\gamma^*}(b) \oplus h_{\gamma^*}(c) = 1$ and still $i \oplus j \oplus l = 0$. Then, as above, we can check that $((a, h_{\gamma^*}(a,p_1) \oplus i),(b,h_{\gamma^*}(b,p_2) \oplus j),(c,h_{\gamma^*}(c,p_3) \oplus l))$ is in $R_1^{A_1}$. The cases in the other direction are easily checked as above.

□

The above lemma presents a general plan to generate CFI-like pairs of structures: Take any unsatisfiable system of equations A such that $A \to_k Z2$ for some fixed k. Then, the construction in the lemma gives us a pair of structures, one satisfiable, the other unsatisfiable, such that both are indistinguishable by the logic $\mathcal{C}^k$.

As an application, such CFI-like constructions may present a way to analyse *preservation* properties of $\mathcal{C}^\omega$. A classical result in finite model theory is the *homomorphism preservation theorem* for first-order logic for finite structures by Rossman [89].

Theorem 6.14 ([89])**.** *Let $\mathscr{C}$ be a class of finite structures that is closed under homomorphisms. Then $\mathscr{C}$ is definable in* FO *if and only if it is definable in* $\exists^+$FO*.*

Interestingly, the dichotomy result from Theorem 2.66 that follows the work of [5, 81, 10] can be stated in a very similar fashion. The following is a restatement of the theorem.

Theorem 6.15 ([5, 81, 10])**.** *Let $\mathscr{C}$ be a class of finite structures, such that there exists a relational constraint language Γ with $\mathscr{C} = \mathrm{CSP}(\Gamma)$. Then $\mathscr{C}$ is definable in $\mathcal{C}^\omega$ if and only if its complement is definable in $\exists^+\mathcal{L}^\omega$.*

As the complement of any CSP(Γ) is closed under homomorphisms, we can see the above result as a special case of a different homomorphism preservation property, namely one for the logic $\mathcal{C}^\omega$ and its fragment $\exists^+\mathcal{L}^\omega$. This motivates the question: Is there way to generalise Theorem 6.15 to obtain a homomorphism preservation theorem for the logic $\mathcal{C}^k$? Or, rephrased, is the following conjecture true?

Conjecture 6.16. *Let $\mathscr{C}$ be a class of finite structures that is closed under homomorphisms. Then $\mathscr{C}$ is definable in $\mathcal{C}^\omega$ if and only if it is definable in $\exists^+\mathcal{L}^\omega$.*

Note that one direction of the claim is trivial: It is known that classes definable in $\exists^+\mathcal{L}^\omega$ are in fact always closed under homomorphisms. Settling this conjecture could involve CFI-like constructions as above. More precisely, in order to prove the claim, it would suffice to prove the following generalisation of Lemma 6.13.

Conjecture 6.17. *For every value l there is a k, such that if for two structures A and B it holds that $A \to_l B$, then there are structures A' and B', such that $A' \equiv^{\mathrm{C}^k} B'$, and $A \to_{\mathrm{hom}} A'$, and $B' \to_{\mathrm{hom}} B$.*

The reasoning is as follows. Suppose there is a class $\mathscr{C}$ that is closed under homomorphisms, definable in $\mathcal{C}^k$ for some fixed k, but not definable in $\exists^+\mathcal{L}^l$ for any l. Then, for every l, there must exist witnesses for the indefinability, namely structures A and B, with $A \to^l B$, but $A \in \mathscr{C}$ and $B \notin \mathscr{C}$. However, if we can find A' and B' as above, then this would imply $A' \in \mathscr{C}$, (since $A \to A'$, and $\mathscr{C}$ is closed under homomorphisms), that $B' \in \mathscr{C}$ (since $A' \equiv^{\mathrm{C}^k} B'$, and $\mathscr{C}$ is definable in $\mathcal{C}^k$), and finally that $B \in \mathscr{C}$ (since $B' \to B$, and $\mathscr{C}$ is closed under homomorphisms), which is a contradiction to our assumption. In turn, this means that every homomorphism closed class $\mathscr{C}$ that is definable in $\mathcal{C}^k$ is also definable in $\exists^+\mathcal{L}^l$ for some l, confirming Conjecture 6.16.

We see that settling the question of whether a homomorphism preservation theorem as in Conjecture 6.16 holds can be reduced to finding a generalised CFI-like construction as specified by Conjecture 6.17. In fact, Lemma 6.13 presents a special case of the construction needed: It proves that Conjecture 6.17 holds when fixing B to a specific structure, namely $Z2$. It is yet unclear whether this construction can be sufficiently generalised to confirm the conjecture for other structures. For instance, a seemingly approachable next step would be to confirm the conjecture when, say, fixing B to the undirected triangle in the vocabulary of undirected graphs. However, this remains an open problem.

6.3 Discussion

In this chapter, we introduced a new technical framework to describe winning strategies in pebble games. More specifically, we formalised strategies for Duplicator in the existential k-pebble game on two structures A and B as mappings between the *k-unravelling* of A onto B. Using this formalism, we were able to show that certain basic properties of the homomorphism relation $\to_{\mathrm{hom}}$ carries over in a more local sense to the relation $\to_k$. In particular, we established a correspondence between injective (and surjective) homomorphisms and injective (and surjective) winning strategies, and showed that an analogue to the classical homomorphism-isomorphism result in the local case. Recently, in an extension of the present work, Abramsky et al. [1] showed that this formalism leads to a natural formulation in category theory, showing that there is yet another angle of view on this subject. In their work, the authors also provide a novel characterisation of the *treewidth* of a structure A in terms of the existence of a homomorphism from A to its k-unravelling.

While the results presented here are still basic, they open up several possible directions of further development. The relations $\to_k$ and $\equiv^{\mathrm{C}^k}$ are often seen as local approximations of homomorphism $\to_{\mathrm{hom}}$ and isomorphism $\simeq_{\mathrm{iso}}$ respectively, and we have shown that this analogy holds up to certain basic properties, such as

the homomorphism-isomorphism theorem. This suggests that other basic notions from the theory of homomorphism may also have a local analogue. For instance, is there a suitable definition of a localised *core* of a structure A?

In a similar spirit, both $\rightarrow_k$ and $\equiv^{\mathrm{C}^k}$ have algorithmic counterparts that serve as approximative homomorphism or isomorphism tests. In the case of $\rightarrow_k$, this corresponds to local consistency algorithms for CSPs. The relation $\equiv^{\mathrm{C}^k}$ is closely related to the equivalence classes created by the k-dimensional Weisfeiler-Leman algorithm. In both cases, there are other algorithms that are either better suitable for a different class of instances (such as few subpowers algorithms [33] for CSPs), or generally more refined (such as [39] for isomorphism). It would be interesting to see if and how these algorithms could fit into this framework, possibly defining new versions of pebble games.

As discussed, another possible future direction is suggested by the formulation of the CFI-construction in terms of strategies. The key technique in our construction of $\mathcal{C}^k$-indistinguishable structures A_0 and A_1 was the translation of a winning strategy for the existential k-pebble game $A \rightarrow_k Z2$ into a winning strategy for the k-pebble bijection game between A_0 and A_1. Note that this was possible for the fixed structure $Z2$. However, it is conceivable that there could be a more general formulation of the present proof that allows us to replace the structure $Z2$ with any other structure of our choice. A positive answer would lead to a homomorphism preservation theorem for the logic $\mathcal{C}^k$, similar in spirit to Rossman's classical result [89].

Chapter 7

Conclusion

In this thesis we studied the framework of constraint problems under various aspects, all related to definability in logic. Here, we recall our key results and provide a glimpse into possible future directions.

A key open problem in the study of constraint problems is to resolve the Feder-Vardi dichotomy conjecture, which claims that for every constraint language Γ, CSP(Γ) is either polynomial time tractable or NP-complete. The work in this thesis was not an attempt to tackle this question directly, but instead aimed to establish similar dichotomies for constraint problems from a descriptive complexity perspective.

For instance, we showed that in the case of Boolean constraint satisfaction, Schaefer's dichotomy theorem [92] can be framed as a kind of definability result. Namely, we showed that the class of tractable problems coincides exactly with those that are definable in fixed-point logic with rank (FPR). This relied on the fact that tractable Boolean CSPs are either solvable using local consistency methods, which can be formulated in Datalog, or can be encoded as the solvability problem for systems of linear equations over the Boolean domain, which is expressible in FPR by design. While this correspondence between algorithmic tractability and definability in FPR is simple and elegant in the Boolean case, it is not known whether this generalises to more general domains, and we pose this as an open problem (see Conjecture 3.14).

Although the Boolean case does not generalise easily, we still established several definability results for the general framework of constraint problems. Perhaps more importantly, we showed that the algebraic approach that led to many key complexity results for CSPs can also be leveraged to obtain corresponding definability results. In the most general framework we consider, namely that of general-valued CSPs (VCSPs), we showed for instance that if two constraint languages Γ and Δ share the same weighted relational clone, then VCSP(Γ) and VCSP(Δ) are interreducible with respect to reductions in fixed-point logic with counting (FPC). This, and similar reducibility results that were previously only known with respect to polynomial time reductions, allowed us to lift previous results from [5] to prove a definability

dichotomy for general VCSPs: Either VCSP(Γ) is definable in FPC, or it is not definable in infinitary logic with counting, $\mathcal{C}^\omega$. Note that unlike in the Boolean CSP case, this dichotomy does not align with the computational complexity of VCSPs: There are VCSP(Γ) that are polynomial time tractable, but not definable in FPC. As expressed earlier, it would be interesting to explore in the future whether these tractable cases could be defined in the stronger logic FPR.

The general framework of VCSPs not only includes classical CSPs, but also optimisation problems. Algorithmically, linear or semidefinite programs (LPs and SDPs) have played a key role in studying these optimisation problems. We showed, by extending ideas from [3, 4], that it is possible to express near-optimal solutions to explicitly given SDPs in the logic FPC. In application to VCSPs, this allowed us to formulate yet another dichotomy, this time with respect to how efficiently certain VCSPs can be solved by the Lasserre relaxation hierarchy [82]. In particular, we proved that for finite-valued constraint languages Γ, either VCSP(Γ) is solved by the first level of the hierarchy, or it requires $\Omega(n)$ levels. This result was achieved by considering a novel complexity measure, the so-called counting width of a class $\mathscr{C}$, $\nu_{\mathscr{C}}(n)$, which denotes the least number k such that the subclass of $\mathscr{C}$ consisting of structures of size at most n is definable in $\mathcal{C}^k$. We then showed that there is a linear relationship between the counting width of a class and the least number of levels in the Lasserre hierarchy required to solve it. We think that this line of reasoning could also generalise to optimisation problems outside the framework of VCSPs, and it is conceivable one could obtain similar lower bound results by studying the counting width of other classes.

From a slightly different angle, we studied the connection between constraint problems and pebble games, a key notion in finite model theory. The idea of using pebble games to study CSPs is not new. Kolaitis and Vardi [75] showed initially that an instance I of CSP(Γ) is accepted by the local consistency algorithm of width k if and only if there is a winning strategy for Duplicator in the existential k-pebble game played on the structures $\mathbf{I}$ and $\mathbf{\Gamma}$. In this thesis, we introduced a new formalism to describe winning strategies in the k-existential pebble game and showed that this formalism can be used to express equivalence conditions between the k-existential pebble game and the k-bijection game. As a result, we obtain a novel way to construct pairs of non-isomorphic structures that are however indistinguishable in the logic $\mathcal{C}^\omega$, reminiscent of the constructions by Cai, Fürer, and Immerman [26].

We hope to have demonstrated that techniques from finite model theory can be powerful tools to improve our understanding on CSPs. As this thesis is the first systematic exploration of the descriptive complexity of constraint problems, we think that many of the results here can be improved in the future.

Particularly intriguing is the question of what role FPR plays in the overall picture. In the Boolean domain, all tractable cases are solved by either local consistency or Gaussian elimination. In logic, the former can be expressed in Datalog, and the latter in FPR. However, while algorithmically, Gaussian elimination has been successfully generalised to capture a large class of CSPs on arbitrary domains,

it is not clear how these more general algorithms can be formulated in FPR. As FPR is one of the current candidate logics to capture P, considering its expressive power in the CSP framework would help to either solidify or refute its claim.

A related direction is to explore other logics in the context of constraint problems, such as *choiceless polynomial time* (CPT), first introduced by Blass et al. [16]. In this thesis, we established the necessary groundwork to leverage the algebraic theory of constraint problems for definability results, and showed for instance that bounded width VCSPs are characterised by their definability in FPC. However, one of the key road blocks we encountered for characterising wider classes in logic is that the expressive power of FPC is limited, while stronger logics such as FPR and CPT are not as well understood. We argue that the framework of constraint problems could be a good proving grounds to study these logics, due to the rich algebraic theory begind it, and the existing foundations connecting this algebraic theory to logic.

Another open question is whether the definability dichotomy for relational CSPs is indicative of a more general preservation theorem between the logics $\exists^+\mathcal{L}^\omega$ and $\mathcal{C}^\omega$, in the style of Rossman's homomorphism preservation theorem [89]. In fact, as discussed in Chapter 6, the definability dichotomy can be seen as a kind of preservation theorem for classes that can be expressed as $\mathrm{CSP}(\Gamma)$ for some constraint language Γ. Since complements of $\mathrm{CSP}(\Gamma)$ are special cases of homomorphism-closed clases, it remains to be seen whether this can be generalised to a proper homomorphism preservation theorem.

Bibliography

[1] Samson Abramsky, Anuj Dawar, and Pengming Wang. The pebbling comonad in finite model theory. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–12, 2017.

[2] Miklós Ajtai. Recursive construction for 3-regular expanders. *Combinatorica*, 14(4):379–416, 1994.

[3] Matthew Anderson, Anuj Dawar, and Bjarki Holm. Maximum matching and linear programming in fixed-point logic with counting. In *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 173–182, 2013.

[4] Matthew Anderson, Anuj Dawar, and Bjarki Holm. Solving linear programs without breaking abstractions. *J. ACM*, 62(6):48:1–48:26, 2015.

[5] Albert Atserias, Andrei A. Bulatov, and Anuj Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410(18):1666–1683, 2009.

[6] Albert Atserias and Elitza N. Maneva. Graph isomorphism, Sherali-Adams relaxations and expressibility in counting logics. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:77, 2011.

[7] Libor Barto. The dichotomy for conservative constraint satisfaction problems revisited. In *Proceedings of the 26th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 301–310, 2011.

[8] Libor Barto. The constraint satisfaction problem and universal algebra. *The Bulletin of Symbolic Logic*, 21(3):319–337, 2015.

[9] Libor Barto. The collapse of the bounded width hierarchy. *Journal of Logic and Computation*, 26(3):923, 2016.

[10] Libor Barto and Marcin Kozik. Constraint satisfaction problems of bounded width. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 595–603, 2009.

[11] Libor Barto, Marcin Kozik, and Todd Niven. The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of Bang-Jensen and Hell). *SIAM Journal on Computing*, 38(5):1782–1802, 2009.

[12] Libor Barto, Andrei A. Krokhin, and Ross Willard. Polymorphisms, and how to use them. In *The Constraint Satisfaction Problem: Complexity and Approximability*, pages 1–44. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2017.

[13] Christoph Berkholz and Martin Grohe. Limitations of algebraic approaches to graph isomorphism testing. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming*, pages 155–166, 2015.

[14] Christoph Berkholz and Oleg Verbitsky. On the speed of constraint propagation and the time complexity of arc consistency testing. In *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science*, pages 159–170, 2013.

[15] Joel Berman, Pawel Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Varieties with few subalgebras of powers. *Transactions of the American Mathematical Society*, 362(3):1445–1473, 2010.

[16] Andreas Blass, Yuri Gurevich, and Saharon Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100(1):141–187, 1999.

[17] Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *J. ACM*, 57(2):9:1–9:41, 2010.

[18] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[19] Andrei Bulatov. Mal'tsev constraints are tractable. *Electronic Colloquium on Computational Complexity (ECCC)*, (034), 2002.

[20] Andrei A. Bulatov. A dichotomy theorem for constraints on a three-element set. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 649–658, 2002.

[21] Andrei A. Bulatov. Tractable conservative constraint satisfaction problems. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, 2003.

[22] Andrei A. Bulatov. Conservative constraint satisfaction re-revisited. *Journal of Computer and System Sciences*, 82(2):347–356, 2016.

[23] Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. *CoRR*, abs/1703.03021, 2017.

[24] Andrei A. Bulatov and Víctor Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM J. Comput.*, 36(1):16–27, 2006.

[25] Andrei A. Bulatov, Peter Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005.

[26] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. Technical report, University of Massachusetts, Amherst, MA, USA, 1991.

[27] Ashok Chandra and David Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25(1):99 – 128, 1982.

[28] Hubie Chen. A rendezvous of logic, complexity, and algebra. *ACM Computing Surveys*, 42(1):2:1–2:32, 2009.

[29] Eden Chlamtac and Madhur Tulsiani. *Convex Relaxations and Integrality Gaps*, pages 139–169. Springer US, Boston, MA, 2012.

[30] David A. Cohen, Martin C. Cooper, Páidí Creed, Peter Jeavons, and Stanislav Živný. An algebraic theory of complexity for discrete optimization. *SIAM Journal on Computing*, 42(5):1915–1939, 2013.

[31] David A. Cohen, Martin C. Cooper, Peter G. Jeavons, and Andrei A. Krokhin. The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170(11):983 – 1016, 2006.

[32] David A. Cohen, Páidí Creed, Peter G. Jeavons, and Stanislav Živný. An algebraic theory of complexity for valued constraints: Establishing a Galois connection. In *Mathematical Foundations of Computer Science 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 231–242. Springer Berlin Heidelberg, 2011.

[33] Víctor Dalmau. Generalized majority-minority operations are tractable. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, pages 438–447, 2005.

[34] Víctor Dalmau. Linear datalog and bounded path duality of relational structures. *Logical Methods in Computer Science*, 1, 2005.

[35] Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 310–326. Springer Berlin Heidelberg, 2002.

[36] Anuj Dawar. On the descriptive complexity of linear algebra. In *Proceedings of the 15th International Workshop on Logic, Language, Information and Computation*, pages 17–25, 2008.

[37] Anuj Dawar, Erich Grädel, Bjarki Holm, Eryk Kopczynski, and Wied Pakusa. Definability of linear equation systems over groups and rings. In *Proceedings of the 21st EACSL Annual Conference on Computer Science Logic*, pages 213–227, 2012.

[38] Anuj Dawar, Martin Grohe, Bjarki Holm, and Bastian Laubner. Logics with rank operators. In *Proceedings of the 24th IEEE Symposium on Logic in Computer Science*, pages 113–122, 2009.

[39] Anuj Dawar and Bjarki Holm. Tractable approximations of graph isomorphism. Algebra, Logic and Algorithms Seminar, Leeds, 2014. Algebra, Logic and Algorithms Seminar, Leeds, 12 November 2014.

[40] Anuj Dawar and Pengming Wang. A definability dichotomy for finite valued CSPs. In *Proceedings of the 24th EACSL Annual Conference on Computer Science Logic*, pages 60–77, 2015.

[41] Anuj Dawar and Pengming Wang. Definability of semidefinite programming and lasserre lower bounds for csps. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–12, 2017.

[42] Wenceslas Fernandez de la Vega and Claire Kenyon-Mathieu. Linear programming relaxations of maxcut. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 53–61, 2007.

[43] Rina Dechter and Daniel Frost. Backjump-based backtracking for constraint satisfaction problems. *Artificial Intelligence*, 136(2):147 – 188, 2002.

[44] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.

[45] László Egri, Benoit Larose, and Pascal Tesson. Symmetric datalog and constraint satisfaction problems in logspace. In *Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 193–202, 2007.

[46] Mária Ercsey-Ravasz and Zoltán Toroczkai. Optimization hardness as transient chaos in an analog approach to constraint satisfaction. *Nature Physics*, 7:966–970, 2011.

[47] Ronald Fagin. *Contributions to the Model Theory of Finite Structures*. PhD thesis, U. C. Berkeley, 1973.

[48] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Comput*, 28:57–104, 1998.

[49] Tomás Feder and Moshe Y. Vardi. Homomorphism closed vs. existential positive. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, 2003.

[50] Eugene C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21(11):958–966, 1978.

[51] Peter Fulla and Stanislav Živný. A Galois connection for valued constraint languages of infinite size. In *Automata, Languages, and Programming*, volume 9134 of *Lecture Notes in Computer Science*, pages 517–528. Springer Berlin Heidelberg, 2015.

[52] Erich Grädel and Martin Otto. Inductive definability with counting on finite structures. In *Proceedings of the 6th Workshop on Computer Science Logic*, 1992.

[53] Erich Grädel and Wied Pakusa. Rank logic is dead, long live rank logic! In *Proceedings of the 24th EACSL Annual Conference on Computer Science Logic*, volume 41, pages 390–404, 2015.

[54] Dima Grigoriev. Complexity of Positivstellensatz proofs for the knapsack. *Computational Complexity*, 10(2):139–154, 2001.

[55] Martin Grohe. Fixed-point logics on planar graphs. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, 1998.

[56] Martin Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. In *Proceedings of the 25h Annual IEEE Symposium on Logic in Computer Science*, pages 179–188, 2010.

[57] Martin Grohe and Julian Mariño. Definability and descriptive complexity on databases of bounded tree-width. In *Proceedings of the 7th International Conference on Database Theory*, pages 70–82, 1999.

[58] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[59] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

[60] Yuri Gurevich. *Logic and the Challenge of Computer Science*. Computer Science Press, 1988.

[61] Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990.

[62] Pavol Hell and Jaroslav Nešetřil. *Graphs and Homomorphisms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2004.

[63] Lauri Hella. Logical hierarchies in PTIME. *Information and Computation*, 129(1):1–19, 1996.

[64] David Hobby and Ralph McKenzie. *The structure of finite algebras*. Contemporary mathematics. American Mathematical Society, 1988.

[65] Bjarki Holm. *Descriptive Complexity Of Linear Algebra*. PhD thesis, University of Cambridge, 2010.

[66] Pawel Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Tractability and learnability arising from algebras with few subpowers. In *Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 213–224, 2007.

[67] Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1):86 – 104, 1986.

[68] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.

[69] Peter Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200(1-2):185–204, 1998.

[70] Leonid G. Khachiyan and Michael J. Todd. On the complexity of approximating the maximal inscribed ellipsoid for a polytope. *Mathematical Programming*, 61(1):137–159, 1993.

[71] Subhash Khot. On the unique games conjecture (invited survey). In *Proceedings of the 25th IEEE Annual Conference on Computational Complexity*, pages 99–121, 2010.

[72] Phokion G. Kolaitis and Madhukar N. Thakur. Logical definability of NP optimization problems. *Information and Computation*, 115(2):321 – 353, 1994.

[73] Phokion G. Kolaitis and Moshe Y. Vardi. Fixpoint logic vs. infinitary logic in finite-model theory. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, pages 46–57, 1992.

[74] Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2):302 – 332, 2000.

[75] Phokion G. Kolaitis and Moshe Y. Vardi. A game-theoretic approach to constraint satisfaction. In *Proceedings of the 17th National Conference on Artificial Intelligence*, 2000.

[76] Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolínek. The complexity of general-valued CSPs. In *Proceedings of the 56th IEEE Annual Symposium on Foundations of Computer Science*, pages 1246–1258, 2015.

[77] Vladimir Kolmogorov and Stanislav Živný. The complexity of conservative valued CSPs. *J. ACM*, 60(2):10:1–10:38, 2013.

[78] Marcin Kozik and Joanna Ochremiak. Algebraic properties of valued constraint satisfaction problem. In *Automata, Languages, and Programming*, volume 9134 of *Lecture Notes in Computer Science*, pages 846–858. Springer Berlin Heidelberg, 2015.

[79] Andrei A. Krokhin and Stanislav Živný. The complexity of valued CSPs. In *The Constraint Satisfaction Problem: Complexity and Approximability*, number 7, pages 233–266. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2017.

[80] Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975.

[81] Benoit Larose and László Zádori. Bounded width problems and algebras. *Algebra universalis*, 56(3-4):439–466, 2007.

[82] Jean B. Lasserre. An explicit exact SDP relaxation for nonlinear 0–1 programs. In *Integer Programming and Combinatorial Optimization*, volume 2081 of *Lecture Notes in Computer Science*, pages 293–303. Springer Berlin Heidelberg, 2001.

[83] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[84] Lászlo Lovász and Alexander Schrijver. Cones of matrices and set-functions and 0–1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991.

[85] Prabhu Manyem. Polynomial-time maximisation classes: Syntactic hierarchy. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(082), 2006.

[86] Martin Otto. *Bounded variable logics and counting — A study in finite models*, volume 9. Springer, 1997.

[87] Arash Rafiey, Jeff Kinne, and Tomás Feder. Dichotomy for digraph homomorphism problems. *CoRR*, abs/1701.02409, 2017.

[88] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., 2006.

[89] Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3):15:1–15:53, 2008.

[90] Thomas Rothvoß. The Lasserre hierarchy in approximation algorithms. In *Lecture Notes for the MAPSP 2013 Tutorial*, 2013.

[91] Francesco Scarcello, Georg Gottlob, and Gianluigi Greco. Uniform constraint satisfaction problems and database theory. In *Complexity of Constraints*, pages 156–195. Springer-Verlag, Berlin, Heidelberg, 2008.

[92] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.

[93] Grant Schoenebeck. Linear level Lasserre lower bounds for certain k-CSPs. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 593–602, 2008.

[94] Hanif D. Sherali and Warren P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

[95] Johan Thapper and Stanislav Živný. The power of linear programming for valued CSPs. In *Proceedings of the 53rd Annual IEEE Symposium on the Foundations of Computer Science*, pages 669–678, 2012.

[96] Johan Thapper and Stanislav Živný. The complexity of finite-valued CSPs. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, pages 695–704, 2013.

[97] Johan Thapper and Stanislav Živný. The limits of SDP relaxations for general-valued CSPs. In *Proceedings of the 32th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2017.

[98] Johan Thapper and Stanislav Zivny. Sherali-Adams relaxations for valued CSPs. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming*, pages 1058–1069, 2015.

[99] Iannis Tourlakis. New lower bounds for vertex cover in the Lovasz-Schrijver hierarchy. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity*, pages 170–182, 2006.

[100] Moshe Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th Annual ACM Symposium on the Theory of Computing*, pages 137–146, 1982.

[101] Boris Weisfeiler and Andrei A. Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

[102] Dmitriy Zhuk. The proof of CSP dichotomy conjecture. *CoRR*, abs/1704.01914, 2017.