THE COMPUTATIONAL COMPLEXITY OF QUANTIFIED CONSTRAINT SATISFACTION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Hubert Ming Chen

August 2004

THE COMPUTATIONAL COMPLEXITY OF QUANTIFIED CONSTRAINT SATISFACTION

Hubert Ming Chen, Ph.D.

Cornell University 2004

The *constraint satisfaction problem (CSP)* is a framework for modelling search problems. An instance of the CSP consists of a set of variables and a set of constraints on the variables; the question is to decide whether or not there is an assignment to the variables satisfying all of the constraints. The *quantified constraint satisfaction problem (QCSP)* is a generalization of the CSP in which variables may be both universally and existentially quantified. The general intractability of the CSP and QCSP motivates the search for restricted cases of these problems that are polynomial-time tractable.

In this dissertation, we investigate the computational complexity of cases of the QCSP where the types of constraints that may appear are restricted. One of our primary tools is the algebraic approach to studying CSP complexity, which can also be used to study QCSP complexity. We first present a pair of new QCSP tractability results; one of these tractability results is arrived at by developing a sound and complete proof system for QCSPs having a certain form. We then introduce a new concept for proving QCSP tractability results called *collapsibility*. The key idea behind collapsibility is that for certain cases of the QCSP, deciding an instance can be reduced to deciding an ensemble of instances, all of which have a bounded number of universally quantified variables and are derived from the original instance by "collapsing" together universally quantified variables. Collapsibility provides a uniform proof technique for deriving QCSP

tractability results which we use both to give alternative proofs of the initial pair of tractability results, as well as to give further tractability results.

Biographical Sketch

Hubert Chen was born at the age of zero. He received a B.S. in Mathematics from Stanford University in June 1998, and expects to receive a Ph.D. in Computer Science from Cornell University in August 2004.

Acknowledgements

First, I wish to thank my advisor Dexter Kozen. Over my years at Cornell, he has been a phenomenal source of guidance, knowledge, and patience; in addition to having learned a lot from him research-wise, his expository ability has greatly influenced and inspired me.

I also thank Bart Selman for all of his help. I first met Bart in the summer of 1996 when I was an intern at AT&T labs and his advisee. I'm very grateful to have received his advice since then, and have benefitted from learning about his perspectives on research.

I have had the privilege of collaborating with a number of different people during my time as a graduate student, and wish to thank all of them: Andrei Bulatov, Steve Chong, Víctor Dalmau, Carla Gomes, Martin Pál, Riccardo Pucella, and Bart Selman. Collaborations with these people were fun and educational, and I hope that future collaborations will be equally pleasant. Riccardo Pucella deserves extra thanks for all of his helpful and useful comments on drafts of papers, which have definitely improved my papers and writing ability.

The staff of the computer science department at Cornell have provided me with a lot of cheerful help; in particular, I thank Beth Howard, Stephanie Meik, Kelly Patwell, and Becky Stewart.

I had the opportunity to visit Oxford in the fall of 2003, and thank Pete Jeavons, Andrei Bulatov, and Víctor Dalmau for a smooth, enjoyable, and productive visit. I am also grateful to Eric Allender of Rutgers for many interesting discussions during the summer of 2001.

While an undergraduate at Stanford, I was fortunate to receive guidance from a number of faculty members, including Steve Kerckhoff, who supervised my undergraduate thesis, and Dan Bump.

Finally, I thank my family–my mother, father, and brother–for all of their support.

This work was supported by a NSF Graduate Research Fellowship.

Table of Contents

Chapter 1

Introduction

The more constraints one imposes, the more one frees one's self of the chains that shackle the spirit.

– Igor Stravinsky, Poetics of Music

1.1 The Constraint Satisfaction Problem

The constraint satisfaction problem (CSP) is widely acknowledged as a convenient framework for modelling search problems. An instance of the CSP consists of a set of variables, a domain, and a set of constraints; each constraint consists of a tuple of variables paired with a relation (over the domain) which contains permitted values for the variable tuple. The question is to decide whether or not there is an assignment mapping each variable to a domain element that satisfies all of the constraints. Problems from many different areas of computer science can be formulated naturally as CSPs including vision [44], boolean satisfiability [51], database theory [40], graph coloring [32, 9], algebra [8], temporal reasoning [53], and truth maintenance [25].

Let us begin by looking at two examples of CSPs.

Example 1 *The* CNF-SATISFIABILITY *problem from propositional logic is to decide, given a propositional formula in* conjunctive normal form (CNF)*, whether or not there is an assignment to the variables satisfying all of the clauses. A propositional formula is said to be in CNF if it is the conjunction of clauses over propositional variables; a clause is the disjunction of literals, where a literal is either a propositional variable or its negation.*

The following is an example instance of the CNF-SATISFIABILITY *problem:*

$$(x \vee y \vee \neg z) \wedge (\neg w \vee y \vee \neg z) \wedge (z \vee \neg y \vee \neg w)$$

We can easily reformulate each instance of the CNF-SATISFIABILITY *problem as an instance of the CSP. For example, the given instance of can be formulated as the CSP instance with variables $\{w, x, y, z\}$, domain $\{0, 1\}$, and constraints*

$$\{R_1(z, x, y), R_2(w, z, y), R_2(y, w, z)\}$$

where the relations R_1 and R_2 are defined as follows:

$$R_1 = \{0, 1\}^3 \setminus \{(1, 0, 0)\}$$

$$R_2 = \{0, 1\}^3 \setminus \{(1, 1, 0)\}$$

An assignment $f : \{w, x, y, z\} \to \{0, 1\}$ satisfies the original instance if and only if it satisfies the CSP instance. The particular assignment $g : \{w, x, y, z\} \to \{0, 1\}$ defined by $g(w) = 0$ and $g(x) = g(y) = g(z) = 1$ is a satisfying assignment for both instances. For example, it satisfies the first constraint $R_1(z, x, y)$ in the given set of constraints, because mapping the variable tuple under g yields a tuple in the accompanying relation: $(g(z), g(x), g(y)) \in R_1$.

Example 2 *A* 3-coloring *of an undirected graph is a labelling of each of its vertices from a 3-element color set, say, $\{\mathsf{red}, \mathsf{blue}, \mathsf{yellow}\}$. The* 3-COLORABILITY *problem* *from graph theory is to decide, given an undirected graph, whether it has a 3-coloring that is* proper *in that no two vertices joined by an edge have the same color. Each instance of the* 3-COLORABILITY *problem can also be easily formulated as an instance of the CSP. The vertices of the graph are represented as variables in the CSP, the domain in the CSP is the set of colors* $\{\mathsf{red}, \mathsf{blue}, \mathsf{yellow}\}$*, and each edge forms a constraint that declares that the two vertices joined must have different colors.*

For instance, consider the instance graph $G = (V, E)$ *of* 3-COLORABILITY *with vertex set*

$$V = \{a_1, a_2, b_1, b_2\}$$

and edge set

$$E = \{\{a_1, b_1\}, \{a_1, b_2\}, \{b_1, b_2\}, \{a_2, b_1\}, \{a_2, b_2\}\}.$$

This instance can be formulated as the CSP instance with variables $V = \{a_1, a_2, b_1, b_2\}$*, domain* $\{\mathsf{red}, \mathsf{blue}, \mathsf{yellow}\}$*, and constraints*

$$\{N(a_1, b_1), N(a_1, b_2), N(b_1, b_2), N(a_2, b_1), N(a_2, b_2)\}$$

where N *is the "not equals" relation on the set of colors, that is,*

$$N = \{\mathsf{red}, \mathsf{blue}, \mathsf{yellow}\}^2 \setminus \{(\mathsf{red}, \mathsf{red}), (\mathsf{blue}, \mathsf{blue}), (\mathsf{yellow}, \mathsf{yellow})\}.$$

An assignment $f : V \to \{\mathsf{red}, \mathsf{blue}, \mathsf{yellow}\}$ *is a proper 3-coloring of the graph* $G = (V, E)$ *if and only if it satisfies all of the given constraints. The particular assignment* $g : V \to \{\mathsf{red}, \mathsf{blue}, \mathsf{yellow}\}$ *defined by* $g(a_1) = g(a_2) = \mathsf{red}$*,* $g(b_1) = \mathsf{blue}$*, and* $g(b_2) = \mathsf{yellow}$ *is a proper 3-coloring, and satisfies all of the given constraints.*

The CSP is in the complexity class NP, since an assignment can be represented in polynomial space, and whether or not it satisfies all given constraints can be checked in polynomial time. The previous example indicates that the 3-COLORABILITY problem, which is well-known to be NP-complete, can be reduced to the CSP; hence, the CSP is NP-complete.

1.2 The Quantified Constraint Satisfaction Problem

All of the variables in a CSP can be thought of as being implicitly existentially quantified: the problem involves deciding whether or not there *exists* an assignment to each of the variables such that all given constraints are satisfied. A useful generalization of the CSP is the quantified constraint satisfaction problem (QCSP), where variables may be both existentially and universally quantified. More specifically, an instance of the QCSP consists of a quantified formula, which consists of a quantifier prefix–an ordered list of variables with associated quantifiers–along with a set of constraints.

Example 3 *Let the relation N be defined as in the previous example. The following is an example instance of the QCSP:*

$$\forall a_1 \forall a_2 \exists b_1 \exists b_2 \{N(a_1, b_1), N(a_1, b_2), N(b_1, b_2), N(a_2, b_1), N(a_2, b_2)\}$$

We can view the CSP instance from the previous example as an instance of the QCSP by existentially quantifying all variables:

$$\exists a_1 \exists a_2 \exists b_1 \exists b_2 \{N(a_1, b_1), N(a_1, b_2), N(b_1, b_2), N(a_2, b_1), N(a_2, b_2)\}$$

An instance of the QCSP may be viewed as a game between two players, the *universal player* and the *existential player*. The universal player sets the universally quantified variables and the existential player sets the existentially quantified variables. Variables are set in the order prescribed by the quantifier prefix. The existential player wins if, after all variables have been set, all constraints are satisfied. The formula is true if the existential player can win, no matter how the universal player sets her variables. For instance, let us look at the quantified formula

$$\forall a_1 \forall a_2 \exists b_1 \exists b_2 \{N(a_1, b_1), N(a_1, b_2), N(b_1, b_2), N(a_2, b_1), N(a_2, b_2)\}.$$

Viewing this quantified formula as a game, the universal player first sets a_1 and a_2, and then the existential player then sets b_1 and b_2. Let us suppose that the universal player sets both a_1 and a_2 to yellow. Then, the existential player may set one of b_1, b_2 to red, and the other to blue in order to satisfy all of the constraints, and win. However, suppose that the universal player sets a_1 and a_2 to different values, say yellow and blue. In order to satisfy the two constraints $\{N(a_1, b_1), N(a_1, b_2)\}$, the existential player must set b_2 to red; similarly, in order to satisfy the two constraints $\{N(a_2, b_1), N(a_2, b_2)\}$, the existential player must set b_1 to red. But, when both b_1 and b_2 are set to red, the constraint $\{N(b_1, b_2)\}$ is falsified. We conclude that there is *no* way for the existential player to win if the universal player sets a_1 and a_2 to yellow and blue (or more generally, to two different colors), and hence that the quantified formula is false.

The generality of the QCSP framework permits the modelling of a variety of computational problems that cannot be expressed using the CSP, for instance, problems from the areas of planning [49], game playing [29], and verification [43]. Of course, the relatively higher expressiveness of the QCSP comes at the price of higher complexity: the QCSP is in general complete for the complexity class PSPACE, which is believed to be much larger than NP.

Because the CSP and the QCSP are in their general formulation intractable, research effort has been devoted to identifying solution techniques that are effective in practice, as well as studying restricted versions of the problems in hopes of identifying polynomial-time tractable cases. This dissertation is concerned with the latter line of research. In particular, we are interested in studying the complexity of restricted versions of the

QCSP where the relations that appear must come from a prescribed set called the *constraint language*.

1.3 Complexity Classification

What is complexity classification?

The results presented in this dissertation belong to a line of work that we refer to as *complexity classification*. In complexity classification, one takes a computational problem that is intractable in its general formulation, and defines a parameterized version of the problem. One then attempts to classify the complexity of the problem relative to each possible instantiation of the parameter.

For instance, let Γ be a *constraint language*, by which we simply mean a set of relations, all of which are over the same domain. We can parameterize the CSP problem according to constraint language: define $\mathsf{CSP}(\Gamma)$ to be the restricted version of the CSP where all relations that occur must come from Γ. After performing this parameterization, we are naturally faced with a classification program: *classify the complexity of* $\mathsf{CSP}(\Gamma)$ *for all constraint languages* Γ.

Probably the first and certainly the most famous complexity classification theorem was given by Schaefer in 1978. He classified the complexity of $\mathsf{CSP}(\Gamma)$ for all constraint languages Γ over a two-element domain.

Theorem 4 *(Schaefer's theorem [51]) Let Γ be a finite constraint language containing relations that are over the two-element domain $\{0, 1\}$. The problem $\mathsf{CSP}(\Gamma)$ is in P if Γ satisfies one of the six conditions below; otherwise, $\mathsf{CSP}(\Gamma)$ is NP-complete.*

1. *Every relation in Γ contains the all-0 tuple $(0, \ldots, 0)$.*

2. *Every relation in Γ contains the all-1 tuple $(1, \ldots, 1)$.*

3. *Every relation in Γ is equivalent to the conjunction of Horn clauses.*

4. *Every relation in Γ is equivalent to the conjunction of dual Horn clauses.*

5. *Every relation in Γ is equivalent to a 2-CNF formula, that is, a conjunction of clauses having size less than or equal to 2.*

6. *Every relation in Γ is equivalent to the conjunction of linear equations of the form $v_1 \oplus \cdots \oplus v_n = c$ where the v_i are variables, $c \in \{0, 1\}$ is a constant, and $\oplus$ denotes the exclusive OR operation.*

Note that we restrict attention to non-empty relations, since any CSP containing an empty relation is trivially unsatisfiable. Also, recall that a (dual) Horn clause is a clause with at most one positive (negative) literal.

Let us look at the six tractable classes of $\mathsf{CSP}(\Gamma)$ problems given by Schaefer's theorem. The first two have a trivial algorithm, which is to just report "satisfiable". This algorithm is correct because the assignment mapping every variable to 0 (respectively, 1) satisfies all constraints. The third class is essentially equivalent and reducible to HORN SATISFIABILITY, which is well-known to be tractable [26], and the fourth is dual to the third: the roles of 0 and 1 are interchanged. The fifth is essentially equivalent and reducible to 2-SATISFIABILITY, which is also well-known to be tractable [1].[1] Finally, the sixth can be viewed as the problem of solving a system of linear equations over the two-element field; this problem is tractable via the Gaussian elimination algorithm.

We thus see that all of the tractable classes given are either trivial, or well-known. Impressively, Schaefer's theorem implies that *any* problem of the form $\mathsf{CSP}(\Gamma)$ where

[1]HORN SATISFIABILITY is the problem of deciding the satisfiability of a conjunction of Horn clauses, and 2-SATISFIABILITY is the problem of deciding the satisfiability of a 2-CNF formula. Both of these problems are subproblems of the CNF-SATISFIABILITY problem.

Γ does not satisfy one of the six stated criteria is NP-complete.

Example 5 *Let T be the relation $\{(1,0,0),(0,1,0),(0,0,1)\}$. The problem $\mathsf{CSP}(\{T\})$ is known as* ONE-IN-THREE SATISFIABILITY*: each constraint $T(v_1,v_2,v_3)$ mandates that exactly one of the three variables $\{v_1,v_2,v_3\}$ be set to true. It can be verified that $\{T\}$ does not satisfy any of the six criteria given by Schaefer's theorem, and so* $\mathsf{CSP}(\{T\})$ *is NP-complete.*

Why complexity classifi cation?

We would now like to articulate why we believe that pursuing complexity classification theorems like Schaefer's is an interesting and useful endeavor. Similar justifications have been offered by other authors [51, 20], and we certainly do not claim any of our discussion to be original.

First, a well-chosen parameterization of a problem can offer a unified framework, in which one can systematically study particular cases of the problem that have been independently investigated. As we have seen, the framework of problems having the form $\mathsf{CSP}(\Gamma)$ includes the algebraic problem of deciding if a system of equations over the field of two elements has a solution, as well as the 2-SATISFIABILITY and HORN SATISFIABILITY problems, two of the best known and oldest tractable versions of the CNF-SATISFIABILITY problem. The $\mathsf{CSP}(\Gamma)$ framework also includes other problems that have been independently investigated in other contexts. For example, it includes the H-colorability problem (for each fixed graph H) from graph theory; the H-colorability problem is to decide if there is a graph homomorphism from an input graph G to the graph H [32].

Although there are always many different ways to parameterize a general computational problem, a complete classification of a problem with respect to a useful parameter provides a very convenient tool for performing complexity analysis. Suppose that we are confronted with a restricted version of the CSP over a two-element domain, and wish to know whether or not the version is tractable. With Schaefer's theorem in hand, we can immediately determine whether or not tractability of the restricted version can be derived from the constraint language; if not, we know that any proof of tractability must crucially use other features of the restricted version. There are examples where problems arising in different areas of computer science have been directly shown to be tractable via formulation as a known tractable $\mathsf{CSP}(\Gamma)$ problem [48, 17]. Moreover, the crisp tractability and intractability results provided by Schaefer's and other classification theorems can be a very convenient starting point for deriving further tractability and intractability results.

Complexity classification results also yield a justification for the scheme of complexity classes that has been developed. One of the great successes of complexity theory thus far has been its ability to characterize practically all "naturally occurring" problems as being either tractable or *complete* for a complexity class. For instance, almost all studied problems in the class NP have been demonstrated to be either in P or NP-complete.[2] That is, the class NP seems to exhibit a form of dichotomy behavior: natural problems inside of it tend to be either easy (in P) or maximally hard for it (NP-complete). That NP exhibits this sort of dichotomy can certainly not be taken for granted, in light of Ladner's theorem, which implies (assuming that P does not equal NP) the existence of problems in NP of *intermediate complexity*: problems that are outside of P, but not NP-complete. Proving classification results like Schaefer's that demonstrate all problems of a large family to be either tractable or complete for a known complexity class, provides

[2]Probably the most prominent counterexample to this trend is the GRAPH ISOMORPHISM problem, which indeed is known to be in NP, but not known to be in P nor NP-complete. There is evidence that it is not NP-complete [52].

validation for the existing scheme of complexity classes: these classification results suggest that there are no "natural" complexity classes intermediate between existing ones, that still need to be defined and studied. Indeed, if one agrees that (for instance) all problems of the form $\mathsf{CSP}(\Gamma)$ are "natural", then a proof that all problems of the form $\mathsf{CSP}(\Gamma)$ are either in P or NP-complete can be viewed as evidence for the claim that *all* "natural" NP problems are either in P or NP-complete–a claim that is probably not readily formalizable, if at all.

An Algebraic Approach

This dissertation is concerned with classifying problems of the form $\mathsf{QCSP}(\Gamma)$, defined analogously to $\mathsf{CSP}(\Gamma)$: for a constraint language Γ, the $\mathsf{QCSP}(\Gamma)$ problem is the restricted version of the QCSP where all relations that occur must come from Γ.

In recent years, a powerful algebraic approach to classifying $\mathsf{CSP}(\Gamma)$ problems has been developed; this approach can be used to study many problem families that are parameterized by constraint language, including $\mathsf{QCSP}(\Gamma)$ problems. We will use this algebraic approach in a central manner to study the problems $\mathsf{QCSP}(\Gamma)$, and now provide a brief overview of it.

A key feature of this approach is that it allows us to associate to every constraint language Γ a set of operations called the *polymorphisms* of Γ having the property that the complexity of $\mathsf{CSP}(\Gamma)$ (and likewise, $\mathsf{QCSP}(\Gamma)$) depends *only* on the polymorphisms of Γ. This makes it possible to attack our complexity classification program by studying sets of polymorphisms, which are dual to constraint languages (sets of relations), instead of constraint languages themselves. This dual viewpoint permits the use of many ideas from algebra to be used towards our quest of complexity classification.

Let us be more precise. Suppose that $R \subseteq D^m$ is a relation, that is, a set of m-tuples over a domain D. We say that an operation $f : D^k \to D$ on D is a *polymorphism* of R if for any choice of (not necessarily distinct) tuples $\overline{t_1}, \ldots, \overline{t_k}$ from R, the length m tuple obtained by applying f in a pointwise manner to the tuples $\overline{t_1}, \ldots, \overline{t_k}$ is also contained in R.

Example 6 *Let R be the relation $\{(1,0),(0,1),(1,1)\}$ over the two-element domain $D = \{0,1\}$. Let $\mu : D^3 \to D$ be the majority function on $\{0,1\}$, that is, the function that is equal to 0 if either two or three of its arguments is equal to 0, and equal to 1 if either two or three of its arguments is equal to 1. We claim that μ is a polymorphism of R. To show this, we need to show that for any choice of tuples $\overline{t_1}, \overline{t_2}, \overline{t_3} \in R$, the tuple $\mu(\overline{t_1}, \overline{t_2}, \overline{t_3})$ is also in R; here, $\mu(\overline{t_1}, \overline{t_2}, \overline{t_3})$ denotes the tuple obtained by applying μ pointwise to the given tuples.*

Let us first consider $\overline{t_1} = (1,0)$, $\overline{t_2} = (0,1)$, and $\overline{t_3} = (1,1)$. We have

$$\begin{aligned} \mu(\overline{t_1}, \overline{t_2}, \overline{t_3}) &= \mu((1,0),(0,1),(1,1)) \\ &= (\mu(1,0,1), \mu(0,1,1)) \\ &= (1,1) \end{aligned}$$

Hence, for this particular choice of $\overline{t_1}, \overline{t_2}, \overline{t_3}$, we have $\mu(\overline{t_1}, \overline{t_2}, \overline{t_3}) \in R$. Since μ is a symmetric function, we have $\mu(\overline{t_1}, \overline{t_2}, \overline{t_3}) = (1,1)$ whenever the tuples $\overline{t_1}, \overline{t_2}, \overline{t_3}$ are different tuples from R.

It can be seen from the definition of μ that when two (or three) of the tuples $\overline{t_1}, \overline{t_2}, \overline{t_3}$ are equal, applying μ to them results in the "equal" tuple, and hence a tuple that is in R presuming that the $\overline{t_i}$ were in R. For example, let us take $\overline{t_1} = (0,1)$ and $\overline{t_2} = \overline{t_3} = (1,0)$.

We have

$$\begin{aligned}\mu(\overline{t_1}, \overline{t_2}, \overline{t_3}) &= \mu((0,1),(1,0),(1,0)) \\ &= (\mu(0,1,1), \mu(1,0,0)) \\ &= (1,0)\end{aligned}$$

We conclude that μ is a polymorphism of R, as desired. In fact, it can be verified that μ is a polymorphism of any arity 2 relation $R \subseteq D^2$ over the two-element domain $D = \{0,1\}$.

As we mentioned, a key fact concerning polymorphisms that we will make use of is that any two constraint languages Γ_1, Γ_2 having exactly the same polymorphisms have exactly the same complexity in that $\mathsf{CSP}(\Gamma_1)$ and $\mathsf{CSP}(\Gamma_2)$ are reducible to each other (and likewise for $\mathsf{QCSP}(\Gamma_1)$ and $\mathsf{QCSP}(\Gamma_2)$). Put succinctly, complexity is an invariant of polymorphisms.

It is consequently possible to give an algebraic formulation of Schaefer's theorem. In particular, each of the tractability criteria has an equivalent formulation in terms of the presence of a polymorphism.

Theorem 7 *[33] A relation R over domain $\{0,1\}$ is equivalent to the conjunction of (dual) Horn clauses if and only if the boolean AND $\wedge$ (respectively, boolean OR $\vee$) function is a polymorphism of R.*

Theorem 8 *[51] A relation R over domain $\{0,1\}$ is equivalent to a 2-CNF formula if and only if the majority function μ is a polymorphism of R.*

Theorem 9 *[51] A relation R over domain $\{0,1\}$ is equivalent to the conjunction of linear equations of the form $v_1 \oplus \cdots \oplus v_n = c$ if and only if the function $x \oplus y \oplus z$ is a polymorphism of R.*

For each of these theorems, the “if” direction is straightforward to check; the reader may find it instructive to work out verifications. Full proofs of the theorems can be found in the monograph [20].

With these theorems along with the easy observation that a relation contains the all-0 (all-1) tuple if and only if it has the constant function 0 (1) as polymorphism, we can reformulate Schaefer’s theorem (Theorem 4) as follows. Let us say that a function f is a polymorphism of a constraint language Γ if it is a polymorphism of every relation $R \in \Gamma$.

Theorem 10 *(Schaefer’s theorem - algebraic formulation) Let Γ be a finite constraint language containing relations that are over the two-element domain $\{0, 1\}$. The problem* $\mathsf{CSP}(\Gamma)$ *is in P if Γ satisfies one of the six conditions below; otherwise,* $\mathsf{CSP}(\Gamma)$ *is NP-complete.*

1. *The constant function 0 is a polymorphism of Γ.*

2. *The constant function 1 is a polymorphism of Γ.*

3. *The boolean AND $\wedge$ function is a polymorphism of Γ.*

4. *The boolean OR $\vee$ function is a polymorphism of Γ.*

5. *The majority function μ is a polymorphism of Γ.*

6. *The function $x \oplus y \oplus z$ is a polymorphism of Γ.*

Algebraic proofs of Schaefer’s theorem have been given; see for instance the paper by Jeavons [34] or the survey [4].

As an example of the utility of the algebraic approach to CSP complexity, we point out that the algebraic formulation of Schaefer’s theorem allows one to transparently see that there is a polynomial-time algorithm for the *meta-question* of deciding, given a constraint language Γ over $\{0, 1\}$, whether or not $\mathsf{CSP}(\Gamma)$ is in P.[3] One polynomial-time algorithm for the meta-question is to simply check, for each of the six polymorphisms f in the statement of Theorem 10, whether or not all relations in Γ have f as polymorphism. The question of how hard it is to determine, for each of the six conditions of Schaefer's theorem, whether an input relation satisfies the condition, was in fact raised as an open question in Schaefer's original paper [51].

1.4 Previous Work

We now survey previous work relevant to our endeavor, with a focus on the work that is most directly comparable to our results.

We first mention that although this dissertation is solely concerned with complexity classification via constraint language parameterization, there have been complexity classification results that use other forms of parameterization. As examples, we name a classification theorem on the *directed subgraph homeomorphism problem* due to Fortune, Hopcroft, and Wyllie [28], classification results on problems concerning circuits built from a fixed set of boolean functions [3], and the study of constraint satisfaction problems based on restricting the interaction among variables [30, 23, 31, 41].

The past decade has seen many complexity classification results of problem families parameterized by constraint language; as we have indicated, study of this parameterization has its origins in Schaefer's 1978 paper [51]. For instance, there has been research on the *counting constraint satisfaction problem* by Creignou and Hermann [19] and Bulatov and Dalmau [10], approximability in boolean constraint satisfaction by Khanna, Sudan, Trevisan, and Williamson [38], *inverse satisfiability* by Kavvadias and

[3]This is observed, for instance, in the monograph [20].

Sideri [37], and minimal model checking by Kirousis and Kolaitis [39]. We point the reader to the surveys [4, 42] and the monograph [20] for further examples.

In the realm of CSP complexity classification based on constraint language, recent years have seen impressive progress on the research program of extending Schaefer's theorem on constraint languages over a two-element domain to a classification theorem for all constraint languages over a finite domain. Feder and Vardi [27] made use of expressibility in Datalog and group theory to describe and explain a number of tractability results. Jeavons [34] established that polymorphisms could be used to approach the CSP classification program; Jeavons, Cohen, and Gyssens [46] gave sufficient conditions for CSP tractability as well as sufficient conditions for CSP intractability described using polymorphisms. For instance, they showed that any constraint language having a semilattice polymorphism is tractable. Bulatov, Krokhin and Jeavons [14] demonstrated that ideas from universal algebra can be used to study CSP complexity. Jeavons, Cohen, and Cooper [35] demonstrated that any constraint language having a near-unanimity polymorphism is tractable. Many further CSP tractability results have been achieved, for example, those by Dalmau and Pearson [24], Dalmau [22], Bulatov, Krokhin, and Jeavons [15], Bulatov [8], Bulatov [6], Bulatov and Jeavons [13], and Chen and Dalmau [18]. Particularly important for our purposes are the papers of Bulatov [8, 6] showing the tractability of Maltsev polymorphisms and 2-semilattice polymorphisms, respectively. Bulatov has also given a complete CSP complexity classification of constraint languages over a three-element domain [7], as well as a classification of *conservative* constraint languages [9].

While there has been a lot of work on CSP complexity classification, there has been much less work on QCSP complexity classification. In domain size two, a classification theorem for quantified formulas with constants was claimed without proof by

Schaefer in his 1978 paper [51]. The polynomial-time tractability of QUANTIFIED 2-SATISFIABILITY was proved by Aspvall, Plass, and Tarjan [1]. A cubic time algorithm for and hence the polynomial-time tractability of QUANTIFIED HORN SATISFIABILITY was demonstrated by Karpinski, Kleine Büning, and Schmitt [36]; a different algorithm with an improved time bound was given by Kleine Büning, Karpinski, and Flögel [16]. Dalmau [21] proved the following dichotomy theorem for quantified formulas analogous to Schaefer's theorem.

Theorem 11 *Let Γ be a finite constraint language containing relations over the two-element domain $\{0, 1\}$. The problem $\mathsf{QCSP}(\Gamma)$ is in P if Γ satisfies one of the conditions 3, 4, 5, 6 of Schaefer's Theorem; otherwise, $\mathsf{QCSP}(\Gamma)$ is PSPACE-complete.*

A proof of Theorem 11 is also given in the monograph [20]. Note that Theorem 11 subsumes the theorem on QCSP claimed by Schaefer, since quantified formulas with constants over a constraint language Γ (with domain $\{0, 1\}$) may be simulated by quantified formulas over the constraint language $\Gamma \cup \{\{(0)\}, \{(1)\}\}$.[4]

Börner, Bulatov, Krokhin, and Jeavons have studied QCSP complexity in domains of arbitrary finite size [5]; their results include the identification of a new Galois connection relevant to QCSP complexity; the tractability of Maltsev polymorphisms and dual discriminator polymorphisms; and, a classification result for constraint languages containing all graphs of permutations. (A dual discriminator polymorphism is a particular type of near-unanimity polymorphism.)

[4]Interestingly, Theorem 11 implies that $\mathsf{QCSP}(\Gamma)$ and $\mathsf{QCSP}(\Gamma \cup \{\{(0)\}, \{(1)\}\})$ are of the same complexity, since for any of the conditions 3, 4, 5, 6 of Schaefer's theorem, a constraint language Γ with domain $\{0, 1\}$ satisfies the condition if and only if $\Gamma \cup \{\{(0)\}, \{(1)\}\}$ does.

1.5 Overview and Contributions

This dissertation is motivated by the research program of understanding and classifying the complexity of $\mathsf{QCSP}(\Gamma)$ for all constraint languages Γ over a finite size domain. The particular contributions of this dissertation are as follows.

In Chapter 3, we prove a pair of QCSP tractability results, namely, the tractability of constraint languages having a near-unanimity polymorphism, and the tractability of constraint languages having a certain type of semilattice polymorphism. These results generalize and unify two different groups of previously established results. Our results broaden the tractability of QUANTIFIED 2-SATISFIABILITY [1] and QUANTIFIED HORN SATISFIABILITY [36, 16] by showing that the constraint languages of these problems are merely instances of families of tractable constraint languages containing constraint languages over domains of all sizes. In addition, near-unanimity and semilattice polymorphisms have been shown to guarantee tractability in the CSP setting [35, 46], and hence our results are QCSP generalizations of known CSP tractability results.

Our proof that near-unanimity polymorphisms are QCSP tractable makes use of algebraic machinery developed by Jeavons, Cohen, and Cooper [35]. Our study of semilattice polymorphisms is centered around a proof system for QCSP instances having semilattice polymorphisms, which we demonstrate to be sound and complete. Our primary use for this proof system is in the development of an algorithm for semilattice polymorphisms, although we believe that it is of independent interest. This proof system and the algorithm for semilattice polymorphisms are inspired by the proof system and algorithm for QUANTIFIED HORN SATISFIABILITY given by Büning, Karpinski, and Flögel [16].

In Chapter 4, we introduce a new concept for proving QCSP tractability results, *collapsibility*. The key insight behind this concept is that for certain problems of the form $\mathsf{QCSP}(\Gamma)$, deciding truth of an instance can be reduced to deciding the truth of an ensemble of instances, all of which have a bounded number of universally quantified variables and are derived from the original instance by "collapsing" together universally quantified variables. This concept provides a uniform proof technique for deriving the tractability of many constraint languages. In particular, this technique reconciles and reveals common structure among the initial two tractable classes of the QCSP that we show to be tractable in Chapter 3, and, as we show, can also be used to derive the QCSP tractability of Maltsev polymorphisms.

In Chapter 5, we study constraint languages having a 2-semilattice polymorphism; 2-semilattice operations generalize semilattice operations. We fully classify 2-semilattice polymorphisms in the QCSP, showing that some such polymorphisms guarantee QCSP tractability, while others do not. This result contrasts with Bulatov's result that all 2-semilattice polymorphisms guarantee CSP tractability [6], and is especially intriguing because it reveals constraint languages whose CSP complexity and QCSP complexity differ. Our positive tractability results make use of and validate the collapsibility machinery developed in Chapter 4.

We have attempted to make the presentation as self-contained as possible, although familiarity with the notion of polynomial-time computation is assumed. Aquaintance with the complexity classes NP, coNP, and PSPACE and the associated completeness notions is useful for appreciating the results, but is almost everywhere not strictly necessary. We name the books [2, 45, 54] as standard texts for these topics.

Chapter 2

Preliminaries

We will occasionally use this arrow notation unless there is danger of no confusion.

– Ronald Graham, Rudiments of Ramsey Theory

We use $[n]$ to denote the set containing the first n positive integers, that is, $\{1, \ldots, n\}$.

For a function $f : A \to B$, we denote by $f[a \to b]$ the extension of f mapping a to b, and we denote by $f|_{A'}$ the restriction of f to a subset $A' \subseteq A$. For a set F of functions $f : A \to B$ and a subset $A' \subseteq A$, we denote by $F|_{A'}$ the set of functions $\{f|_{A'} : f \in F\}$.

A *tuple* over a set S is an element of S^k (for some $k \geq 1$), and is said to have arity k. The ith coordinate of a tuple $\overline{t}$ is denoted by t_i. A *relation* over a set S is a subset of S^k (for some $k \geq 1$), and is said to have arity k.

2.1 Constraint Satisfaction

Intuitively, the constraint satisfaction problem involves deciding if there is a mapping from a set of variables to a domain satisfying given constraints. We will only be concerned with finite-domain constraint satisfaction, and hence adopt the convention that a *domain* is a nonempty set of finite size. Throughout, we will use D to denote the domain of constraints and constraint networks.

Constraints. A *constraint* consists of two parts: a scope, which consists of variables, and a collection of values that the scope may take on. For ease of notation, we will use two different notions of constraint. Throughout this dissertation, we will freely interchange between these two different notions.

The first notion of constraint uses a relation to specify the allowable values for the scope. A *relation-based constraint* is an expression of the form $R(\overline{v})$, where R is a relation (over a domain D) and $\overline{v}$ is a tuple of variables such that R and $\overline{v}$ have the same arity. The tuple of variables $\overline{v}$ is called the *scope* of the constraint $R(\overline{v})$. A constraint $R(\overline{v})$ of arity $k \geq 1$ is *satisfied* by a mapping $f : V \rightarrow D$ defined on the variables in $\overline{v} = (v_1, \ldots, v_k)$ if $(f(v_1), \ldots, f(v_k)) \in R$.

The second notion of constraint uses a set of functions to specify the allowable values for the scope. A *function-based constraint* is a pair $\langle S, R\rangle$, where S is a set of variables, and R is a set of functions, all of which map from S to the same domain D. The set of variables S is called the *scope* of the constraint $\langle S, R\rangle$. A constraint $\langle S, R\rangle$ is *satisfied* by a mapping $f : V \rightarrow D$ defined on the variables in S if $f|_S \in R$.

Every relation-based constraint naturally induces a function-based constraint, and vice-versa. The relation-based constraint $R(v_1, \ldots, v_k)$ induces the function-based constraint $\langle\{v_1, \ldots, v_k\}, R'\rangle$ where R' contains all functions $f : \{v_1, \ldots, v_k\} \rightarrow D$ such that f satisfies $R(v_1, \ldots, v_k)$. The function-based constraint $\langle S, R\rangle$ induces the relation-based constraint $R'(v_1, \ldots, v_k)$ where $v_1, \ldots, v_k$ are distinct variables having the property that $\{v_1, \ldots, v_k\} = S$, and $R' = \{(f(v_1), \ldots, f(v_k)) \mid f \in R\}$. It is straightforward to verify that a mapping $f : V \rightarrow D$ satisfies a relation-based constraint $R(\overline{v})$ if and only if it satisfies the function-based constraint induced by $R(\overline{v})$; similarly, a mapping $f : V \to D$ satisfies a function-based constraint $\langle S, R\rangle$ if and only if it satisfies the relation-based constraint induced by $\langle S, R\rangle$.

Constraint networks. A *constraint network* is a finite set of constraints, all of which have relation over the same domain, and is said to be over the variable set V if all of its constraints have variables from V. A mapping $f : V \to D$ is a *solution* or *satisfying assignment* of a constraint network $\mathcal{C}$ (over V) if it satisfies all constraints in $\mathcal{C}$.

We formally define the constraint satisfaction problem as follows.

Defi nition 12 *The* constraint satisfaction problem, *denoted by* CSP*, is the problem of deciding, given a constraint network* $\mathcal{C}$ *over variable set* V *and domain* D*, whether or not there exists a satisfying assignment* $f : V \to D$ *of* $\mathcal{C}$*.*

It will be useful to consider the restriction of a constraint network to a subset of the variable set. Let $\mathcal{C}$ be a constraint network over variable set V and let U be a subset of V. Define $\mathcal{C}|_U$ to be the constraint network $\{C|_U \mid C \in \mathcal{C}\}$ where for a constraint $C = \langle S, R\rangle$, we use $C|_U$ to denote the constraint $\langle S \cap U, R|_{S\cap U}\rangle$.

2.2 Quantified Constraint Satisfaction

In the constraint satisfaction problem, one can think of the variables as being implicitly existentially quantified: the problem involves deciding whether or not there *exists* an assignment to each of the variables such that all given constraints are satisfied. The quantified constraint satisfaction problem (QCSP) is more general, in that universal quantification–in addition to existential quantification–of variables is permitted.

In the terminology of logic, an instance of the QCSP will be a closed, first-order quantified formula consisting of a quantifier prefix followed by a constraint network.

This dissertation will be concerned almost exclusively with formulas of this type; hence, we will for simplicity call such a formula a *quantified formula*, and point out explicitly when free variables are present, as indicated in the following definitions.

Definition 13 *A* quantified formula with free variables W *is an expression of the form* $Q_1 v_1 \dots Q_n v_n \mathcal{C}$*, where each* Q_i *is a quantifier from the set* $\{\forall, \exists\}$*, the variables* v_i *are distinct and not included in* W*, and* $\mathcal{C}$ *is a constraint network over the variable set* $\{v_1, \dots, v_n\} \cup W$*.*

Definition 14 *A quantified formula having no free variables is referred to as a* quantified formula.

We define truth of quantified formulas (with free variables) inductively on the number of bound variables, just as in first-order logic.

Definition 15 *Let* $\phi = Q_1 v_1 \dots Q_n v_n \mathcal{C}$ *be a quantified formula with free variables* W*. The formula* ϕ *is true relative to an assignment* $f : W \to D$ *if and only if:*

- *there are no bound variables (that is,* $n = 0$*) and* f *satisfies* $\mathcal{C}$*,*
- $Q_1 = \exists$ *and there exists an element* $d \in D$ *such that the formula* $Q_2 v_2 \dots Q_n v_n \mathcal{C}$ *is true relative to the assignment* $f[v_1 \to d]$*, or*
- $Q_1 = \forall$ *and for all elements* $d \in D$*, the formula* $Q_2 v_2 \dots Q_n v_n \mathcal{C}$ *is true relative to the assignment* $f[v_1 \to d]$*.*

A quantified formula (without free variables) is considered to be true if it is true relative to the empty assignment.

Definition 16 *The* quantified constraint satisfaction problem*, denoted by* QCSP*, is the problem of deciding, given a quantified formula* ϕ*, whether or not* ϕ *is true.*

It can be seen that the restricted version of the QCSP problem where all quantifiers must be existential, is equivalent to the CSP problem.

Quantifier prefixes. When $\phi = Qv_1 \ldots Q_n v_n \mathcal{C}$ is a quantified formula, we call $Q_1 v_1 \ldots Q_n v_n$ the *quantifier prefix* of ϕ. When the quantifier Q_i associated with a variable v_i is existential (that is, $Q_i = \exists$), we say that v_i is an existentially quantified variable or existential variable; similarly, when Q_i is universal (that is, $Q_i = \forall$), we say that v_i is an universally quantified variable or universal variable. We let V_ϕ, Y_ϕ, and X_ϕ denote the variables, universally quantified variables, and existentially quantified variables of a quantified formula ϕ, respectively; we drop the ϕ subscript when it is understood from the context. Notice that V_ϕ is the disjoint union of Y_ϕ and X_ϕ, as we assumed the variables v_i in the quantifier prefix to be distinct.

In a quantifier prefix $Q_1 v_1 \ldots Q_n v_n$, we say that the variable v_i *comes before* the variable v_j if $i < j$, and that the variable v_i *comes after* the variable v_j if $i > j$. We often assume that the universally quantified variables of a quantified formula are denoted $y_1, \ldots, y_{|Y|}$, where y_i comes before y_j for $i < j$; similarly, we often assume that the existentially quantified variables of a quantified formula are denoted $x_1, \ldots, x_{|X|}$, where x_i comes before x_j for $i < j$.

When W is a non-empty subset of the variable set V_ϕ of a quantified formula ϕ, we use $\mathsf{first}_\phi(W)$ to denote the unique variable in W coming before all of the other variables in W, and we use $\mathsf{last}_\phi(W)$ to denote the unique variable in W coming after all of the other variables in W.

Strategies and truth. We now give a characterization of true quantified formulas that will be particularly wieldy in our investigation. This characterization is based on the notion of *Skolem functions* from logic. To describe the characterization, we use the terms *strategy* and *adversary*: a *strategy* gives a way of setting the existential variables of a quantified formula in reaction to an *adversary* which sets the universal variables. As we discussed in Chapter 1, one way to view a quantified formula is as a game between two players: the universal player, who sets the universal variables, and the existential player, who sets the existential variables. Variables are set in the order prescribed by the quantifier prefix, and the formula is true if the existential player can always ensure that the constraint network is satisfied by the final variable setting. The following characterization of true quantified formulas is a formalization of this view.

Definition 17 *A* strategy *for a quantified formula ϕ is a sequence of mappings*

$$\sigma = \{\sigma_i : D^{u(i)} \rightarrow D\}_{i \in [n]}$$

where $u(i)$ denotes the number of universal variables coming before the ith existential variable x_i.

Note that when $u(i) = 0$, we consider the mapping σ_i to be a constant, that is, an element of D.

Definition 18 *An* adversary *for a quantified formula ϕ is a mapping $\tau : Y_\phi \rightarrow D$.*

When σ is a strategy and τ is an adversary for a quantified formula ϕ, we use (σ, τ) to denote the assignment that results when the universal variables are set according to τ and the existential variables are set according to σ; formally, (σ, τ) is the mapping from V_ϕ to D defined by

$$(\sigma, \tau)(y_i) = \tau(y_i)$$

for $y_i \in Y_\phi$, and

$$(\sigma, \tau)(x_i) = \sigma_i(\tau(y_1), \ldots, \tau(y_{u(i)}))$$

for $x_i \in X_\phi$.

A strategy σ for the quantified formula ϕ is said to be *winning* or a *winning strategy* if for all adversaries τ for ϕ, the assignment $(\sigma, \tau) : V_\phi \to D$ satisfies the constraint network $\mathcal{C}$ of ϕ.

Proposition 19 *A quantified formula ϕ is true if and only if it has a winning strategy.*

2.3 Problem Formulation

This dissertation focuses on a parameterized version of the QCSP that is defined relative to a *constraint language*. A *constraint language* is defined to be a set of relations (not necessarily of the same arity), all of which are over the same domain.

Defi nition 20 *Let Γ be a constraint language. The $\mathsf{QCSP}(\Gamma)$ problem is to decide, given as input a quantified formula ϕ with constraints having relations from Γ, whether or not ϕ is true.*

We define the $\mathsf{CSP}(\Gamma)$ problem to be the restriction of the $\mathsf{QCSP}(\Gamma)$ problem to instances where all quantifiers are existential.

The following research problem has been studied intensely, particularly over the past decade.

Research Problem 1 *Classify the complexity of the problem $\mathsf{CSP}(\Gamma)$ for all constraint languages Γ over a finite domain.*

This dissertation is concerned with the following research problem.

Research Problem 2 *Classify the complexity of the problem $\mathsf{QCSP}(\Gamma)$ for all constraint languages Γ over a finite domain.*

In this section, we will show that one only needs to consider constraint languages of a particular form to resolve Research Problems 1 and 2.

We say that a decision problem is *tractable* if it is decidable in polynomial time. We will say that a constraint language Γ is *tractable* (or, for emphasis, *QCSP tractable*) if $\mathsf{QCSP}(\Gamma)$ is tractable. We will say that a constraint language Γ is *CSP tractable* if $\mathsf{CSP}(\Gamma)$ is tractable. We will say that a constraint language Γ reduces to another constraint language Γ' if $\mathsf{QCSP}(\Gamma)$ reduces to $\mathsf{QCSP}(\Gamma')$ via a polynomial-time many-one reduction–the only form of reduction we will use in this dissertation [2]. All results in this section are either explicit or implicit in the papers [34, 12, 5]; we state them in terms of the QCSP, although they were first observed (and all hold) for the CSP.

We begin with a simple observation: adding relations to a constraint language can only make it harder.

Proposition 21 *Let Γ, Γ' be constraint languages such that $\Gamma \subseteq \Gamma'$. Then Γ reduces to Γ'.*

Proof: The problem $\mathsf{QCSP}(\Gamma)$ reduces to $\mathsf{QCSP}(\Gamma')$ by the identity reduction. •

Defi nition 22 *Let Γ be a constraint language. A relation R of arity k is* expressible *by Γ if there exists a quantified formula ϕ with free variables $\{w_1, \dots, w_k\}$, having only existential quantifiers, and having only relations from $\Gamma \cup \{=_D\}$ such that*

$$R = \{(f(w_1), \dots, f(w_k)) \mid \phi \text{ is true relative to } f : \{w_1, \dots, w_k\} \to D\}.$$

Here, $=_D$ denotes the equality relation on the domain D.

We use $\langle\Gamma\rangle$ to denote the set of all relations expressible by Γ.[1]

[1]Note that our definition of expressibility only permits the use of existential quantification. In [5], the set of all relations expressible by Γ using both existential and universal quantification, denoted by $[\Gamma]$, is defined and studied. The discussion in this section mostly holds for $[\Gamma]$ in place of $\langle\Gamma\rangle$, but none our results require the use of $[\Gamma]$. We refer the interested reader to [5].

We will show in this section that the complexity of a constraint language Γ, in a certain precise sense, depends only on the constraint language $\langle \Gamma \rangle$. Roughly speaking, this is because for a constraint language Γ, we can add to Γ relations that are expressible by Γ without changing the complexity of Γ. In particular, suppose we start with a finite constraint language Γ and add to it relations expressible by Γ to obtain a second finite constraint language Γ'. Even though Γ' is larger than Γ, it is no harder than Γ: it can be seen that Γ' reduces to Γ by the following proposition.

Proposition 23 *Let* Γ *be any constraint language, and let* Γ' *be a finite constraint language such that* $\Gamma' \subseteq \langle \Gamma \rangle$. *Then* Γ' *reduces to* Γ.

Proof idea: The constraint language Γ' reduces to $\Gamma \cup \{=_D\}$ via the reduction that, given an instance ϕ of $\mathsf{QCSP}(\Gamma')$, creates an instance of $\mathsf{QCSP}(\Gamma \cup \{=_D\})$ by replacing each constraint $R(w_1, \ldots, w_k)$ of ϕ with a quantified formula expressing R that is over $\Gamma \cup \{=_D\}$ and has free variables $\{w_1, \ldots, w_k\}$, as in Definition 22. It is straightforward to prove that the problem $\mathsf{QCSP}(\Gamma \cup \{=_D\})$ reduces to $\mathsf{QCSP}(\Gamma)$ [5]. •

Observe that the constraint language $\langle \Gamma \rangle$ is always infinite. We can define a notion of tractability on *infinite* constraint languages so that a constraint language Γ is "tractable" if and only if $\langle \Gamma \rangle$ is.

Defi nition 24 *A constraint language* Γ *is* locally tractable *if for every finite subset* Γ' *of* $\langle \Gamma \rangle$, *the constraint language* Γ' *is tractable. A constraint language* Γ *is* locally intractable *if it is not locally tractable.*

Note that Definition 24 is interesting only for infinite constraint languages: a finite constraint language is tractable if and only if it is locally tractable.

The next proposition shows that, from the standpoint of local tractability, the constraint languages Γ and $\langle \Gamma \rangle$ have the same complexity.

Proposition 25 *A constraint language Γ is locally tractable if and only if $\langle\Gamma\rangle$ is locally tractable.*

Proof: Every finite subset Γ' of $\langle\Gamma\rangle$ reduces to a finite subset of Γ by the reduction of Proposition 23, and so every such finite subset Γ' is locally tractable when Γ is locally tractable. That the local tractability of $\langle\Gamma\rangle$ implies the local tractability of Γ is immediate from the fact that $\Gamma \subseteq \langle\Gamma\rangle$. •

From Proposition 25, we can see that the program of classifying which constraint languages Γ are locally tractable can be "reduced" to the program of classifying which constraint languages of the form $\langle\Gamma\rangle$ are locally tractable.

As was mentioned, a finite constraint language is locally tractable if and only if it is tractable. This fact, however, does not hold (at least *a priori*) in the case of infinite constraint languages. To emphasize the distinction between local tractability and tractability of infinite constraint languages, the term *global tractability* is sometimes employed.

Definition 26 *A constraint language Γ is* globally tractable *if it is tractable, and is* globally intractable *if it is not tractable.*[2]

We remark that, for all studied cases of the CSP and QCSP, infinite constraint languages that are locally tractable are also globally tractable; it has been conjectured that in the CSP setting, local tractability always implies global tractability [12].

As stated in Research Problem 2, we are interested in identifying all constraint languages (both finite and infinite) that are globally tractable. All of the results in this dissertation will take on one of two forms:

[2]When Γ is a finite constraint language, the complexity of $\mathsf{QCSP}(\Gamma)$ is not sensitive to how relations in Γ are represented, so long as each relation has a constant size representation. However, when Γ is an infinite constraint language, one needs to be precise in describing how relations are represented. In global tractability, we assume that relations are represented by an explicit listing of all tuples.

- demonstration that a constraint language of the form $\langle\Gamma\rangle$ is globally tractable, or
- demonstration that a constraint language of the form $\langle\Gamma\rangle$ is locally intractable.

Global tractability implies local tractability; likewise, local intractability implies global intractability, so both our positive and negative results are of the strongest possible type.

Indeed, global tractability of $\langle\Gamma\rangle$ implies global tractability of Γ (since $\Gamma \subseteq \langle\Gamma\rangle$), and local intractability of $\langle\Gamma\rangle$ implies the local (and hence global) intractability of Γ by Proposition 25. Demonstrating that all constraint languages $\langle\Gamma\rangle$ are either globally tractable or locally intractable would therefore resolve Research Problem 2, and so we are justified in focusing attention on constraint languages $\langle\Gamma\rangle$.

2.4 Algebra

It has been shown that powerful algebraic theory can be used to study the complexity of $\mathsf{CSP}(\Gamma)$ problems [46, 34]. The ideas of [46, 34] can be adopted in a straightforward way to also study the complexity of $\mathsf{QCSP}(\Gamma)$ problems from an algebraic viewpoint. This algebraic theory associates to every constraint language Γ a set of operations, called the *polymorphisms* of Γ, which characterizes the constraint language $\langle\Gamma\rangle$ and hence the complexity of $\mathsf{QCSP}(\Gamma)$.

We begin by defining what it means for an operation to be a *polymorphism* of a relation.

Definition 27 *An operation $\mu : D^k \to D$ is a* polymorphism *of a relation $R \subseteq D^m$ if for all tuples $\overline{t_1}, \ldots, \overline{t_k} \in R$, the tuple $(\mu(t_{11}, \ldots, t_{k1}), \ldots, \mu(t_{1m}, \ldots, t_{km}))$ is in R.*

In other words, an operation μ on D of rank k is a polymorphism of a relation R if, for any choice of k tuples $\overline{t_1}, \ldots, \overline{t_k}$ from R, the tuple obtained by acting on the tuples $\overline{t_i}$ in a coordinate-wise manner by μ, is also contained in R.

We extend this definition to constraint languages as follows.

Definition 28 *An operation μ is a polymorphism of a constraint language Γ if μ is a polymorphism of all relations $R \in \Gamma$.*

We will also use the following terminology.

Definition 29 *When μ is a polymorphism of a relation R (or a constraint language Γ), we say that R (or Γ) is* invariant *under μ.*

The set of all polymorphisms of a constraint language Γ, as well as the set of all relations invariant under all operations in a given set, play an important role in our investigation and are thus worthy of their own notation.

Definition 30 *Let $\mathcal{O}_D$ denote the set of all finite rank operations over D, and let $\mathcal{R}_D$ denote the set of all finite arity relations over D.*

When $\Gamma \subseteq \mathcal{R}_D$ is a set of relations (that is, a constraint language), we define

$$\mathsf{Pol}(\Gamma) = \{\mu \in \mathcal{O}_D \mid \mu \text{ is a polymorphism of } \Gamma\}.$$

When $F \subseteq \mathcal{O}_D$ is a set of operations, we define

$$\mathsf{Inv}(F) = \{R \in \mathcal{R}_D \mid R \text{ is invariant under all operations } \mu \in F\}.$$

It is straightforward to verify that any relation expressible by a constraint language Γ possesses the polymorphisms of Γ, that is, $\langle\Gamma\rangle \subseteq \mathsf{Inv}(\mathsf{Pol}(\Gamma))$. In fact, the converse holds as well: any relation invariant under all polymorphisms of Γ is expressible by Γ, that is, $\mathsf{Inv}(\mathsf{Pol}(\Gamma)) \subseteq \langle\Gamma\rangle$.

Theorem 31 *For any constraint language Γ, it holds that $\langle\Gamma\rangle = \mathsf{Inv}(\mathsf{Pol}(\Gamma))$.*

There is a proof of Theorem 31 in the paper [34].

Theorem 31 demonstrates that the set of relations expressible by a constraint language Γ depends *only* on the polymorphisms of Γ: given the polymorphisms of Γ, the set $\langle \Gamma \rangle$ can be computed by an application of the $\mathsf{Inv}(\cdot)$ operator. We showed in the previous section that the complexity of a constraint language Γ can be studied by studying the constraint language $\langle \Gamma \rangle$. Thus, we can approach our complexity classification program by studying sets of *operations*.

Definition 32 *When $F \subseteq \mathcal{O}_D$ is a set of operations, we use $\mathsf{QCSP}(F)$ to denote the problem $\mathsf{QCSP}(\mathsf{Inv}(F))$. When F consists of only one element, that is, $F = \{\mu\}$, we use the notation $\mathsf{QCSP}(\mu)$ instead of $\mathsf{QCSP}(\{\mu\})$.*

We define $\mathsf{CSP}(F)$ and $\mathsf{CSP}(\mu)$ analogously.

With this notation in hand, our research problem can be reduced to the problem of classifying the complexity of $\mathsf{QCSP}(F)$ for all possible sets of polymorphisms F. This is because, by Theorem 31, the problem $\mathsf{QCSP}(\langle \Gamma \rangle)$ is equivalent to the problem $\mathsf{QCSP}(\mathsf{Pol}(\Gamma))$.

We have observed that larger constraint languages are higher in complexity than smaller ones (Proposition 21). The situation for sets of operations is dual to that for constraint languages: larger sets of operations have lower complexity than smaller ones.

Proposition 33 *Let $F, F' \subseteq \mathcal{O}_D$ be sets of operations such that $F \subseteq F'$. Then the problem $\mathsf{QCSP}(F')$ reduces to the problem $\mathsf{QCSP}(F)$.*

Proof: Immediate from Proposition 21 and the fact that $F \subseteq F'$ implies $\mathsf{Inv}(F') \subseteq \mathsf{Inv}(F)$. •

Correspondingly, the literature contains a number of results that show that the presence of a particular type of operation in $\mathsf{Pol}(\Gamma)$ implies the CSP tractability of Γ. That is, there are many results having the form:

"If μ is an operation of type X, and $\mu \in \mathsf{Pol}(\Gamma)$, then $\mathsf{CSP}(\Gamma)$ is tractable."

Such results can be succinctly stated as $\mathsf{CSP}(\mu)$ tractability results, since (1) such a result implies the tractability of $\mathsf{CSP}(\mathsf{Inv}(\mu)) = \mathsf{CSP}(\mu)$, and (2) for any constraint language Γ for which $\mu \in \mathsf{Pol}(\Gamma)$, it holds that $\Gamma \subseteq \mathsf{Inv}(\mu)$ and hence $\mathsf{CSP}(\Gamma)$ is a subproblem of $\mathsf{CSP}(\mu)$.

Some examples of such results are given in the following theorems.

A *semilattice operation* is a binary operation that is associative, commutative, and idempotent.

Theorem 34 *[46] Let $\oplus$ be a semilattice operation on a finite set D. If Γ is a constraint language such that $\oplus \in \mathsf{Pol}(\Gamma)$, then $\mathsf{CSP}(\Gamma)$ is decidable in polynomial time.*

An operation $\mu : D^k \to D$ is *near-unanimity* if it is of arity $k \geq 3$ and it returns the value $d \in D$ whenever at least $k-1$ of its arguments are equal to d.

Theorem 35 *[35] Let μ be a near-unanimity operation on a finite set D. If Γ is a constraint language such that $\mu \in \mathsf{Pol}(\Gamma)$, then $\mathsf{CSP}(\Gamma)$ is decidable in polynomial time.*

A *Maltsev operation* is a ternary operation m satisfying the identities $m(x,x,y) = m(y,x,x) = y$.

Theorem 36 *[8] Let μ be a Maltsev operation on a finite set D. If Γ is a constraint language such that $\mu \in \mathsf{Pol}(\Gamma)$, then $\mathsf{CSP}(\Gamma)$ is decidable in polynomial time.*

The tractability results of Theorems 34, 35, and 36 in fact generalize the non-trivial tractable cases of Schaefer's theorem (see Theorem 10): it can be verified that the boolean AND $\wedge$ and boolean OR $\vee$ functions are semilattice operations, the majority function μ discussed in Chapter 1 is a near-unanimity operation, and the function $x \oplus y \oplus z$ is a Maltsev operation.

We have just seen that the presence of certain types of polymorphisms can imply CSP tractability of a constraint language. On the flip side, it has been shown that constraint languages that are "polymorphism improverished"–lacking non-trivial polymorphisms–are CSP intractable.

Let us say that an operation $f : D^k \rightarrow D$ is a *non-constant essentially unary operation* if there exists $i \in [k]$ and a non-constant function $g : D \rightarrow D$ such that $f(d_1, \ldots, d_k) = g(d_i)$ for all $d_1, \ldots, d_k \in D$.

Theorem 37 *[46] Let Γ be a constraint language. If $\mathsf{Pol}(\Gamma)$ contains only non-constant essentially unary operations, then $\mathsf{CSP}(\Gamma)$ is NP-complete.*

As we have mentioned, our complexity classification program amounts to studying all problems having the form $\mathsf{QCSP}(F)$, where $F = \mathsf{Pol}(\Gamma)$ for a constraint language Γ. Operation sets of the form $\mathsf{Pol}(\Gamma)$ have a particular structure: they are *clones*.

Definition 38 *A set of operations $F \subseteq \mathcal{O}_D$ is a* clone *if:*

- *for all m, n such that $1 \leq m \leq n$, the set F contains the projection operation $\pi_{m,n} : D^n \rightarrow D$ defined by $\pi_{m,n}(x_1, \ldots, x_n) = x_m$, and*
- *the set F is closed under composition, where the composition of an operation $f : D^n \rightarrow D$ and operations $g_1, \ldots, g_n : D^m \rightarrow D$ is the operation $h : D^m \rightarrow D$ defined by $h(x_1, \ldots, x_m) = f(g_1(x_1, \ldots, x_m), \ldots, g_n(x_1, \ldots, x_m))$.*

Proposition 39 *For any constraint language Γ, the set $\mathsf{Pol}(\Gamma)$ is a clone.*

Analogously, relation sets of the form $\mathsf{Inv}(F)$ and $\langle\Gamma\rangle$ have a particular structure: they are *relational clones*, also known as *co-clones*.

Definition 40 *A set of relations* $\Gamma \subseteq \mathcal{R}_D$ *is a* relational clone *if:*

- *the equality relation* $=_D$ *is contained in the set* Γ*,*
- *the set* Γ *is closed under intersection, that is, if* $R, S \in \Gamma$ *are of the same arity, then* $R \cap S \in \Gamma$*,*
- *for any* $R \in \Gamma$ *of arity* k*, the relation*

  $$R \times D = \{(d_1, \ldots, d_{k+1}) \mid (d_1, \ldots, d_k) \in R, d_{k+1} \in D\}$$

  is also in Γ*, and*
- *for any* $R \in \Gamma$ *of arity* k *and any sequence of (not necessarily distinct) indices* $i_1, \ldots, i_m \in [k]$*, the relation* $\{(d_{i_1}, \ldots, d_{i_m}) \mid (d_1, \ldots, d_k) \in R\}$ *is also in* Γ*.*

Proposition 41 *For any set of operations* $F \subseteq \mathcal{O}_D$*, the set* $\mathsf{Inv}(F)$ *is a relational clone.*

Proposition 42 *For any constraint language* Γ*, the set* $\langle\Gamma\rangle$ *is a relational clone; moreover, every relational clone is equal to* $\langle\Gamma\rangle$ *for some constraint language* Γ*.*

Proof: The first part follows immediately from Theorem 31 and Proposition 41. The second part follows from the fact that for any relational clone Γ, it holds that $\Gamma = \langle\Gamma\rangle$. •

We remark that $\langle\Gamma\rangle$ is the smallest relational clone containing Γ.

The operators $\mathsf{Pol}(\cdot)$ and $\mathsf{Inv}(\cdot)$ give a *Galois connection* between the power set of $\mathcal{R}_D$ and the power set of $\mathcal{O}_D$. We also mention that there is a one-to-one correspondence between clones and relational clones, given by the operators $\mathsf{Pol}(\cdot)$ and $\mathsf{Inv}(\cdot)$. We refer the interested reader to the book [47] for more information on these operators.

In the last section, we stated that our results will concern problems of the form $\mathsf{QCSP}(\langle\Gamma\rangle)$. In fact, they are phrased in terms of problems having the form $\mathsf{QCSP}(\mu)$ (for an operation μ), which, by the previous two propositions, are also problems of the form $\mathsf{QCSP}(\langle\Gamma\rangle)$.

We conclude this chapter by giving two auxiliary results that will be useful throughout our study of quantified constraint satisfaction.

Proposition 43 *For every instance ϕ of the* QCSP *problem, there is an equivalent instance of the* CSP *problem having no more than $|X_\phi| \cdot |D|^{|Y_\phi|}$ variables and no more than $c \cdot |D|^{|Y_\phi|}$ constraints, where c is the number of constraints in ϕ.*

Proof: The idea is to create an instance of the CSP where the variables are the possible output values of the mappings of a strategy. The CSP instance created will be satisfiable if and only if there is a winning strategy for the original QCSP instance ϕ.

For each existential variable $x \in X_\phi$ of ϕ, let $Y[x]$ denote the universal variables (in ϕ) coming before x. The variables of the CSP instance are all pairs of the form (x, α), where x is an existential variable and α is a mapping from $Y[x]$ to D. For every constraint $R(\overline{v})$ of ϕ and for every adversary $\tau : Y_\phi \to D$, we create a constraint in the CSP instance as follows. Let us assume, for the sake of notation, that $\overline{v}$ has the form $(x_1, \ldots, x_m, y_1, \ldots, y_n)$, where the x_i are existential variables and the y_i are universal variables; we create a constraint equivalent to

$$R((x_1, \tau|_{Y[x_1]}), \ldots, (x_m, \tau|_{Y[x_m]}), \tau(y_1), \ldots, \tau(y_n))$$

and having scope $((x_1, \tau|_{Y[x_1]}), \ldots, (x_m, \tau|_{Y[x_m]}))$. Note that for a single constraint $R(\overline{v})$, different adversaries may cause the same constraint to be created.

Suppose that f is an assignment to the variables of the CSP instance. It is straightforward to verify that f satisfies the constraints of the CSP instance if and only if the strategy which sets variable $x \in X_\phi$ to $f((x, \tau|_{Y[x]}))$ in reaction to adversary $\tau : Y_\phi \to D$ is a winning strategy. •

An operation $\mu : D^k \to D$ is *idempotent* if $\mu(d, \ldots, d) = d$ for all $d \in D$.

Proposition 44 *Suppose that μ is an idempotent operation, and fix a constant $j \geq 1$. The problem $\mathsf{QCSP}(\mu)$, when restricted to instances with j or fewer universal quantifiers, reduces (via many-one polynomial-time reduction) to $\mathsf{CSP}(\mu)$.*

Proof: The reduction is the translation given in the proof of Proposition 43; this translation can be carried out in time polynomial in the size of the original QCSP instance when the number of universal quantifiers is constant. This translation involves creating relations that are obtained from relations of the QCSP instance by instantiating some coordinate positions with constants. The resulting CSP instance is an instance of $\mathsf{QCSP}(\mu)$ since idempotent polymorphisms are preserved when constant instantiation is performed; this is straightforward to verify. •

Chapter 3

A Pair of Tractability Results

A foolish consistency is the hobgoblin of little minds, adored by little statesmen and philosophers and divines.

– Ralph Waldo Emerson, Self-Reliance

In this chapter, we give a pair of QCSP tractability results: we show that two classes of constraint languages–those having near-unanimity polymorphisms, and those having certain semilattice polymorphisms–are QCSP tractable. In the CSP setting, near-unanimity polymorphisms and semilattice polymorphisms have been shown to guarantee tractability [46, 35]; our QCSP tractability results are thus generalizations of known CSP tractability results. These two classes of polymorphisms are fundamental in the CSP setting. In particular, they arise very naturally in the consideration of *consistency* algorithms for the CSP, algorithms which, roughly speaking, infer new constraints from a given CSP instance by the examination of bounded-size subsets of the variable set.

Near-unanimity polymorphisms characterize exactly those constraint languages in which establishing a sufficiently high level of *local consistency* implies the *global consistency* and hence satisfiability of a CSP instance, as will be discussed in Section 3.1.

Semilattice polymorphisms were one of the first CSP tractable classes identified [46], and are closely tied to the algorithm of establishing *arc consistency*, a very basic algorithm which ensures that every instance of a variable in a CSP instance is "supported" by the same values. Establishing arc consistency is a solution procedure for semilattice polymorphisms; moreover, every constraint language for which establishing arc consistency is a solution procedure can, in a certain precise sense, be explained to be tractable via the presence of a semilattice polymorphism.[1]

Our proof that near-unanimity polymorphisms are QCSP tractable makes use of machinery developed by Jeavons, Cohen, and Cooper [35], who studied near-unanimity polymorphisms in the CSP setting. Our study of semilattice polymorphisms includes the presentation of a proof system for QCSP instances having semilattice polymorphisms, which we demonstrate to be sound and complete, and which we use to develop an algorithm for such instances. Our proof system and algorithm are inspired by the proof system and algorithm for QUANTIFIED HORN SATISFIABILITY given by Büning, Karpinski, and Flögel [16]; our proof system can be viewed as a generalization of theirs.

3.1 Near-Unanimity Polymorphisms

An operation $\mu : D^k \to D$ is said to be *near-unanimity* if it is of arity $k \geq 3$ and it returns the value $d \in D$ whenever at least $k-1$ of its arguments are equal to d; that is, if $\bar{t} \in D^k$ is a k-tuple such that there exists $i \in [k]$ with $t_j = d$ for all $j \in [k] \setminus \{i\}$, then $\mu(\bar{t}) = d$.

[1]In particular, Dalmau and Pearson [24] have shown that, in the CSP setting, a constraint language is solvable via arc consistency if and only if it has a set function polymorphism; such constraint languages, when viewed as relational structures, are homomorphically equivalent to, and hence can be solved by the same algorithms as, constraint languages having a semilattice polymorphism.

We first turn to review the notion of consistency; our presentation is based on that of Jeavons, Cohen, and Cooper [35].

Definition 45 *Let $\mathcal{C}$ be a constraint network over the variable set V.*

- *The constraint network $\mathcal{C}$ is j-*consistent *(where $j \geq 1$) if for every subset V' of V of size $j-1$ and for every variable $v \in V \setminus V'$, any solution $f : V' \to D$ of $\mathcal{C}|_{V'}$ can be extended to a solution $f' : V' \cup \{v\} \to D$ of $\mathcal{C}|_{V' \cup \{v\}}$.*
- *The constraint network $\mathcal{C}$ is* strongly k-consistent *if it is j-consistent for all $j \in \{1, \ldots, k\}$.*
- *The set $\mathcal{C}$ is* globally consistent *if it is strongly $|V|$-consistent.*

Global consistency is an extremely desirable property, as a constraint network that is globally consistent has a solution: this is because 1-consistency implies that for any variable v_1, the constraint network $\mathcal{C}|_{\{v_1\}}$ has a solution; 2-consistency implies that for any variable v_2 not equal to v_1, the constraint network $\mathcal{C}|_{\{v_1,v_2\}}$ has a solution; continuing in this fashion, we can infer that $\mathcal{C}|_V = \mathcal{C}$ has a solution.

Proposition 46 *If a constraint network $\mathcal{C}$ is globally consistent, then it has a solution.*

The general intractability of the CSP implies that it is not possible to efficiently decide if a given constraint network can be made globally consistent. However, it is possible to efficiently convert a given constraint network into one that is either *locally consistent*, or has an empty constraint (and is hence trivially unsatisfiable).[2] By *locally consistent*, we mean strongly k-consistent for a constant k. Consider the following algorithm:

[2]By an *empty constraint*, we mean a constraint having an empty relation and a non-empty scope.

- Introduce for each subset U of the variable set with size k, a constraint $\langle U, R\rangle$ where R contains all assignments to the U.

- For all introduced constraints $\langle U, R\rangle$, do the following: eliminate from R any mappings $f \in R$ such that f does not satisfy $\mathcal{C}|_U$, where $\mathcal{C}$ denotes the entire constraint network (including the newly introduced constraints).

- Repeat the previous step until no changes are made.

It is straightforward to verify that this procedure takes polynomial time, and that when the procedure terminates, the resulting constraint network either contains an empty constraint, or is k-consistent. Moreover, the resulting constraint network has exactly the same solutions as the original.

For some classes of constraint networks, performing the above procedure and then checking for an empty constraint is a correct decision procedure for the class. Indeed, the following theorem shows that when a constraint language is invariant under a near-unanimity operation, ensuring a sufficiently high, but constant, degree of local consistency implies global consistency.

Theorem 47 *[35] Suppose that Γ is a constraint language invariant under a near-unanimity operation of arity k. If a constraint network $\mathcal{C}$ over Γ is strongly k-consistent, then $\mathcal{C}$ is globally consistent.*

Theorem 47 implies that for any near-unanimity operation $\mu : D^k \to D$, the problem $\mathsf{CSP}(\mu)$ is polynomial-time tractable, since (as was discussed) it is possible in polynomial time to transform a constraint network into one that is either strongly k-consistent, or trivially unsatisfiable.

Theorem 48 *[35] Let μ be a near-unanimity operation on a finite set D. The problem $\mathsf{CSP}(\mu)$ is polynomial-time decidable.*

We demonstrate that the CSP tractability result of Theorem 48 can be generalized to the QCSP.

Theorem 49 *Let μ be a near-unanimity operation on a finite set D. The problem* $\mathsf{QCSP}(\mu)$ *is polynomial-time decidable.*

Proof: Let $\mu : D^k \to D$ be a near-unanimity operation of rank k, and let $\phi = Q_1 v_1 \ldots Q_n v_n \mathcal{C}$ be an instance of the $\mathsf{QCSP}(\mu)$ problem. We show that, when $n \geq k$, it is possible to compute from ϕ a new instance ϕ' of the $\mathsf{QCSP}(\mu)$ problem that has one fewer quantifier than ϕ, has no more constraints than ϕ, and is true if and only if ϕ is. It will be possible to compute ϕ' from ϕ in polynomial time, and hence any instance of $\mathsf{QCSP}(\mu)$ can be solved in polynomial time by iteratively eliminating quantifiers to obtain a formula with at most k quantifiers, which can be solved by brute force.

The new formula ϕ' is obtained from ϕ by elimination of the innermost quantifier and associated quantified variable, $Q_n v_n$. We split into two cases depending on the type of Q_n.

- Case $Q_n = \exists$: Obtain the constraint network $\mathcal{C}_0$ from $\mathcal{C}$ by attempting to establish k-consistency on $\mathcal{C}$. Note that attempting to establishing k-consistency can be performed by a sequence of projections and intersections, two operations which preserve polymorphisms, and so we may assume that $\mathcal{C}_0$ is invariant under μ.

  If $\mathcal{C}_0$ has an empty constraint, then ϕ is false. Otherwise, define the constraint network $\mathcal{C}'$ to be $\mathcal{C}_0|_{V \setminus \{v_n\}}$, and define ϕ' to be $Q_1 v_1 \ldots Q_{n-1} v_{n-1} \mathcal{C}'$. Clearly, if ϕ has a winning strategy Σ, then ϕ' also has a winning strategy, namely, the strategy where all existential variables of ϕ' are set as in the strategy Σ. On the other hand, if ϕ' has a winning strategy Σ, then ϕ has a winning strategy that sets the existential variables in $V \setminus \{v_n\}$ according to Σ. Since the constraint network $\mathcal{C}_0$ is strongly k-consistent and hence globally consistent, any solution to $\mathcal{C}' = \mathcal{C}_0|_{V \setminus \{v_n\}}$ can be extended to a solution to $\mathcal{C}_0$ (which has the same solutions as $\mathcal{C}$), and hence Theorem 47 guarantees that v_n can be set.

- Case $Q_n = \forall$: Compute the constraint network $\mathcal{C}' = \{C' \mid C \in \mathcal{C}\}$, where for a constraint $C = \langle S, R\rangle$, the constraint $C' = \langle S', R'\rangle$ is defined by $S' = S \setminus \{v_n\}$ and letting R' contain all mappings $f : S' \to D$ having the property that all extensions $g : S \to D$ of f are contained in R. (We say that $g : S \to D$ is an extension of $f : S' \to D$ if $g|_{S'} = f$.) It is straightforward to verify that $\mathcal{C}'$ is invariant under μ, and that the formulas ϕ and $\phi' = Q_1 v_1 \ldots Q_{n-1} v_{n-1} \mathcal{C}'$ have exactly the same winning strategies.

•

3.2 Semilattice Polymorphisms

A *semilattice operation* is a binary operation that is associative, commutative, and idempotent. It is well-known (and easy to verify) that every semilattice operation $\oplus : D^2 \to D$ induces a partial order $\leq_\oplus$, where $a \leq_\oplus b$ if and only if $a \oplus b = b$ (for all $a, b \in D$).

Semilattice polymorphisms have been shown to guarantee CSP tractability [46].

Theorem 50 *[46] Let $\oplus : D^2 \to D$ be a semilattice operation on a finite set D. The problem* $\mathsf{CSP}(\oplus)$ *is polynomial-time decidable.*

A polynomial-time algorithm for deciding $\mathsf{CSP}(\oplus)$ is to establish *arc consistency* and then to check for an empty constraint. We say that a constraint network $\mathcal{C}$ is *arc consistent* if for any two constraints $\langle S, R\rangle, \langle S', R'\rangle \in \mathcal{C}$ and a variable $v \in S \cap S'$, it holds that $\{f(v) \mid f \in R\} = \{f'(v) \mid f' \in R'\}$. It is straightforward to verify that in polynomial time, one can convert a constraint network into one that is arc consistent. Moreover, this conversion can be done in a way that preserves polymorphisms, as with the local consistency procedure discussed in the previous section. We show that performing this conversion and then checking for an empty constraint is a correct decision procedure for the problems at hand.

Proof idea: Let $\mathcal{C}$ be an arc consistent instance of $\mathsf{CSP}(\oplus)$ that does not contain any empty constraint. We show that $\mathcal{C}$ has a solution $f : V \rightarrow D$. For each variable v of $\mathcal{C}$, pick a constraint $\langle S, R \rangle$ such that $v \in S$ and define $f(v)$ to be the maximal element of $\{g(v) \mid g \in R\}$ with respect to $\leq_{\oplus}$. A maximal element exists because $\oplus$ is a polymorphism of R. Moreover, the definition of $f(v)$ is independent of the constraint $\langle S, R \rangle$ chosen, since $\mathcal{C}$ is assumed to be arc consistent. Using the fact that $\oplus$ is a polymorphism of all constraints in $\mathcal{C}$, it can be shown that f satisfies all constraints in $\mathcal{C}$. •

We say that a semilattice operation $\oplus : D^2 \rightarrow D$ has a *unit element* if there exists an element $u \in D$ such that $u \oplus d = d \oplus u = d$ for all $d \in D$. In this section, we demonstrate that for any semilattice operation $\oplus$ having a unit element, the problem $\mathsf{QCSP}(\oplus)$ is polynomial-time tractable. This tractability result is optimal in that all other remaining semilattice operations–those lacking a unit element–are intractable, as we will demonstrate in Chapter 5. We prove this tractability result in three steps. First, we demonstrate that any constraint invariant under a semilattice operation can be decomposed into the conjunction of Horn clauses of a certain form, which we call *semilattice Horn clauses*. Second, we give a sound and complete proof system for quantified formulas consisting of semilattice Horn clauses. Third, we develop an algorithm for semilattice operations having unit based on the proof system. We remark that the theory developed in the first two steps apply to all semilattice operations, and not just semilattice operations having a unit element.

Our first step is to demonstrate that any constraint invariant under a semilattice operation is equivalent to the conjunction of Horn clauses of a certain form. We introduce the following definitions and notation. Let $\oplus : D^2 \to D$ be a semilattice operation. A *positive literal* is an expression of the form $v \leq_\oplus d$, where v is a variable and $d \in D$; and, an *negative literal* is an expression of the form $v \not\leq_\oplus d$, where v is a variable and $d \in D$. A *semilattice Horn clause* is a disjunction of positive and negative literals that contains at most one positive literal. Throughout this section, we will use the term *Horn clause* as a shorthand for *semilattice Horn clause*.

Theorem 51 *If C a constraint that is invariant under a semilattice operation $\oplus$, then C is logically equivalent to (that is, has the same satisfying assignments as) a conjunction of Horn clauses.*

Proof: Let $C = \langle S, R\rangle$ be a constraint invariant under a semilattice operation $\oplus : D^2 \to D$. It suffices to show that for each function $g : S \to D$ *not* contained in R, there exists a Horn clause H_g such that any mapping $f : S \to D$ in R satisfies H_g, but the mapping g does not satisfy H_g. Fix $g \notin R$, and define $S_g = \{f \in R \mid f \leq_\oplus g\}$ where we extend the partial order $\leq_\oplus$ to mappings in the usual coordinate-wise fashion: $f \leq_\oplus g$ if for all $v \in S$, it holds that $f(v) \leq_\oplus g(v)$.

If S_g is empty, then the Horn clause $H_g = \bigvee_{v \in S}(v \not\leq_\oplus g(v))$ has the desired properties. If S_g is not empty, define the mapping $h : S \to D$ by $h(v) = \oplus_{f \in S_g} f(v)$. Since $\langle S, R\rangle$ is invariant under $\oplus$, it holds that $h \in R$ and indeed that $h \in S_g$. Because $g \notin R$, there exists a variable $v_0 \in S$ such that $g(v_0) \not\leq_\oplus h(v_0)$. The Horn clause

$$H_g = (v_0 \leq_\oplus h(v_0)) \vee \bigvee_{v \in S \setminus \{v_0\}} (v \not\leq_\oplus g(v))$$

has the desired properties. •

Theorem 51 generalizes Theorem 7, which states that the relation of a constraint over domain $\{0, 1\}$ is invariant under boolean AND ($\wedge$) if and only if the constraint is logically equivalent to the conjunction of Horn clauses. To see this, we let $\oplus = \wedge$; then the only non-trivial positive literal is one of the form $v \leq_{\oplus} 1$, which is equivalent to the positive literal v; and the only non-trivial negative literal is one of the form $v \not\leq_{\oplus} 1$, which is equivalent to the negative literal $\overline{v}$. Hence, for $\oplus = \wedge$, semilattice Horn clauses are exactly propositional Horn clauses–disjunctions of literals over boolean variables containing no more than one positive literal.

We now give a proof system, which we call *Q-semilattice-resolution*, that is sound and complete for quantified formulas consisting of semilattice Horn clauses. Our description of the proof system will make use of the following terminology. A literal occurring in a QCSP is an $\exists$-literal ($\forall$-literal) if its variable is a $\exists$-variable ($\forall$-variable). When H is a Horn clause appearing within a quantified formula, we let $H^{\forall}$ denote the set containing all $\forall$-literals in H; and, we let $H^{\exists}$ denote the set containing all $\exists$-literals in H. A Horn clause H appearing in a QCSP ϕ is an *existential unit clause* if it contains only one $\exists$-literal, the single $\exists$-literal is positive, and for every $\forall$-variable y appearing in H, the variable y comes before the variable of the $\exists$-literal in the quantification order of the formula ϕ.

For notational convenience, we will view a semilattice Horn clause as a set (instead of a disjunction) of literals.

Definition 52 *(*Q-semilattice-resolution*) Let ϕ be a quantified formula with constraint network $\mathcal{C}$ consisting of semilattice Horn clauses. A semilattice Horn clause H can be derived from ϕ by* Q-semilattice-resolution *(denoted $\phi \vdash H$) if it can be obtained by one or more applications of the following rules.*

0. For all $H \in \mathcal{C}$, it holds that $\phi \vdash H$.

1. *If $\phi \vdash H_1$, $\phi \vdash H_2$, and there exist an $\exists$-variable x and elements $a, b \in D$ such that $(x \leq_\oplus a) \in H_1$ and $(x \leq_\oplus b) \in H_2$, then*

$$\phi \vdash (H_1 \setminus \{x \leq_\oplus a\}) \cup (H_2 \setminus \{x \leq_\oplus b\}) \cup L$$

where $L = \{x \leq_\oplus (\oplus\{d \in A \mid d \leq_\oplus a, d \leq_\oplus b\})\}$ if $\{d \in A \mid d \leq_\oplus a, d \leq_\oplus b\}$ is non-empty, and $L = \emptyset$ otherwise.

2. *If $\phi \vdash H$ and there exist a domain element $a \in D$ and an $\exists$-variable x such that $(x \leq_\oplus a) \in H$, then*

$$\phi \vdash (H \setminus \{x \leq_\oplus a\}) \cup \{x \leq_\oplus b\}$$

for all $b \in D$ such that $a \leq_\oplus b$.

3. *If $\phi \vdash U$ where U is an existential unit clause with positive literal $(x \leq_\oplus a)$, and $\phi \vdash H$ where $(x \not\leq_\oplus a) \in H$, then*

$$\phi \vdash (U \setminus \{x \leq_\oplus a\}) \cup (H \setminus \{x \not\leq_\oplus a\}).$$

4. *If $\phi \vdash H$, the last variable in the quantification order of ϕ occurring in H is a $\forall$-variable y, and there exists a value $a \in D$ such that the assignment $y = d$ does not satisfy H_y (the clause containing all y-literals in H), then*

$$\phi \vdash H \setminus H_y.$$

We demonstrate this proof system to be sound and complete, as follows.

Theorem 53 *Let ϕ be a quantified formula with constraint network consisting of semi-lattice Horn clauses. The formula ϕ is unsatisfiable if and only if $\phi \vdash \emptyset$ using Q-semilattice-resolution.*

Proof: It is straightforward to prove that the proof system is sound in the sense that if Σ is a winning strategy for ϕ and $\phi \vdash H$, then Σ is a winning strategy for ϕ with H added to its constraint network. We thus prove that if it is not the case that $\phi \vdash \emptyset$, then ϕ is true.

We assume for the sake of notation that ϕ has the form

$$\forall y_1 \exists x_1 \ldots \forall y_n \exists x_n \mathcal{C}$$

where $\mathcal{C}$ consists only of Horn clauses with respect to a semilattice operation $\oplus : D^2 \to D$. When $\overline{a} = (a_1, \ldots, a_k)$ is a k-tuple over D, let $f_{\overline{a}}$ denote the mapping taking y_i to a_i for all $i \in [k]$. For each $k \in [n]$, define the mapping $\sigma_k : D^k \to D$ by

$$\sigma_k(a_1, \ldots, a_k) = \oplus\{d \mid f_{\overline{a}}[x_k = d] \text{ satisfies all clauses in } \mathcal{U}_{\phi,x_k}\}$$

where $\mathcal{U}_{\phi,x_k}$ denotes all existential unit clauses U containing x_k such that $\phi \vdash U$.

We will show that the mappings $\{\sigma_k\}_{k\in[n]}$ are a winning strategy for ϕ. We first show that each mapping σ_k is well-defined, that is, for each k and tuple $\overline{a} \in D^k$, the set

$$\{d \mid f_{\overline{a}}[x_k = d] \text{ satisfies all clauses in } \mathcal{U}_{\phi,x_k}\}$$

is non-empty. We prove this by contradiction; suppose that for some k and tuple $\overline{a} \in D^k$, the set is empty. Then, for every $d \in D$ there exists an existential unit clause U_d containing x_k such that $\phi \vdash U_d$ and $f_{\overline{a}}$ does not satisfy U_d. We have (for all $d \in D$) that $f_{\overline{a}}$ does not satisfy $U_d^{\forall}$, and that $x_k = d$ does not satisfy the single positive literal in $U_d^{\exists}$. Applying rule 1 to all of the U_d, we obtain $\phi \vdash \cup_{d\in D} U_d^{\forall}$. Since $f_{\overline{a}}$ does not satisfy $\cup_{d\in D} U_d^{\forall}$, by repeated application of rule 4, we obtain that $\phi \vdash \emptyset$, contradicting our assumption that it is not the case that $\phi \vdash \emptyset$.

We now show that $\{\sigma_k\}_{k\in[n]}$ is a winning strategy for ϕ. Let $\tau : \{y_1, \ldots, y_n\} \to D$ be an assignment to the $\forall$-variables of ϕ; we wish to show that (σ, τ) satisfies all clauses of ϕ. It suffices to show that (σ, τ) satisfies all clauses H such that $\phi \vdash H$. We prove this by induction on the number t of negative $\exists$-literals that H contains.

We split into three cases.

Case 1: $t = 0$ and H does not contain any $\exists$-literals. If (σ, τ) does not satisfy H, then the empty set can be derived from H using rule 4.

Case 2: $t = 0$ and H contains a positive $\exists$-literal. Obtain H' by applying rule 4 to eliminate $\forall$-literals in H until rule 4 is no longer applicable. If the last variable in H' (relative to the quantification order of ϕ) is a $\forall$-variable y, then for all $a \in D$, every extension of the assignment $y = a$ satisfies H', and so in particular H' is satisfied by (σ, τ). If the last variable in H' is an $\exists$-variable x, then H' is an existential unit clause and is satisfied by (σ, τ) by the definition of the σ_k.

Case 3: H contains an negative $\exists$-literal $x_k \not\leq_\oplus d$. Suppose (for a contradiction) that (σ, τ) does not satisfy H; then $(\sigma, \tau)(x_k) \leq d$. By reasoning similar that used in to the above argument that the mappings σ_i are well-defined, there is a derivable existential unit clause U with literal $x_k \leq_\oplus (\sigma, \tau)(x_k)$ such that (σ, τ) does not satisfy $U \setminus \{x_k \leq_\oplus (\sigma, \tau)(x_k)\}$. Set $U' = (U \setminus \{x \leq_\oplus (\sigma, \tau)(x_k)\}) \cup \{x_k \leq d\}$. By applying rule 2 to U, we obtain $\phi \vdash U'$. Set $H' = (U' \setminus \{x_k \leq_\oplus d\}) \cup (H \setminus \{x_k \not\leq_\oplus d\})$. By rule 3, $\phi \vdash H'$. Notice that H' is not satisfied by (σ, τ); this contradicts our inductive hypothesis, since H' contains one less negative $\exists$-literal than H. •

We can now show that any semilattice operation with unit is QCSP tractable: this is done by using the Q-semilattice-resolution proof system to show that any QCSP instance invariant under such a semilattice operation can be reduced to an ensemble of QCSP instances, each of which has a single universal variable.

Theorem 54 *Let $\oplus : D^2 \to D$ be a semilattice operation with unit element on a finite set D. The problem* $\mathsf{QCSP}(\oplus)$ *is polynomial-time decidable.*

Proof: Let $\phi = \forall y_1 \exists x_1 \ldots \forall y_n \exists x_n \mathcal{C}$ be an instance of the $\mathsf{QCSP}(\oplus)$ problem, and let $\oplus$ be a semilattice operation with unit element u. Notice that for all elements $d \in D$, it holds that $u \leq_\oplus d$. By appeal to Theorem 51, there is a constraint network $\mathcal{H}$ containing only Horn clauses that is logically equivalent to $\mathcal{C}$.

Let $D^\exists$ denote the subset of $\mathcal{H}$ containing all clauses with a positive $\exists$-literal; Let $D^\forall$ denote the subset of $\mathcal{H}$ containing all clauses with a positive $\forall$-literal; and let U denote the subset of $\mathcal{H}$ containing all clauses having only negative literals.

It can be verified that every clause derivable from ϕ by Q-semilattice-resolution is derivable from

$$\phi_H \stackrel{\text{def}}{=} \forall y_1 \exists x_1 \ldots \forall y_n \exists x_n (D^\exists \cup \{H\})$$

for some $H \in D^\forall \cup U$. This is because when H is a clause in $D^\forall \cup U$, H cannot be combined with any other clause in $D^\forall \cup U$ to derive a further clause, in Q-semilattice-resolution.

It follows that $\phi \vdash \emptyset$ if and only if there exists $H \in D^\forall \cup U$ such that $\phi_H \vdash \emptyset$. Hence, deciding whether or not ϕ is true thus amounts to deciding whether or not ϕ_H is true, for all $H \in D^\forall \cup U$. When $H \in U$, the formula ϕ_H is true if and only if $\phi_H[y_1 = \cdots = y_n = u]$ (that is, the formula ϕ where all $\forall$-variables have been replaced with the unit element u) is true: this is because the only universal literals in ϕ_H are negative, and so any strategy that wins against the adversary setting all y_i to u is a winning strategy. When $H \in D^\forall$, let y denote the variable in the positive $\forall$-literal of H (which is the only positive $\forall$-literal in ϕ_H); similar reasoning shows that ϕ_H is true if and only if ϕ'_H is true, where ϕ'_H is obtained from ϕ_H by setting all $\forall$-variables except for y to the unit element u.

Hence, the formula ϕ is true if and only if each of the formulas $\phi_1, \ldots, \phi_n$ is true, where ϕ_i is obtained from ϕ by setting all $\forall$-variables except for y_i to the unit element

u. By appeal to Proposition 44, deciding the truth of an input instance ϕ of $\mathsf{QCSP}(\oplus)$ can be performed in polynomial time. $\bullet$

Chapter 4

Collapsibility

In this chapter, we introduce a new concept for use with QCSPs which we call *collapsibility*. The key insight behind this concept is that for certain problems of the form $\mathsf{QCSP}(\Gamma)$, deciding truth of an instance can be reduced to deciding the truth of an ensemble of instances, all of which have a bounded number of universally quantified variables and are derived from the original instance by "collapsing" together universally quantified variables.

More specifically, we will define a j-collapsing of a quantified formula ϕ to be any quantified formula obtained by choosing j of the universally quantified variables in ϕ, and associating together the remaining universally quantified variables (in a particular way to be made precise). It is always the case that when ϕ is a true quantified formula, any j-collapsing of ϕ is also true. We will show that certain problems of the form $\mathsf{QCSP}(\Gamma)$ are *j-collapsible*, for some constant j, in that the *converse* also holds: an instance of $\mathsf{QCSP}(\Gamma)$ is true if all j-collapsings of ϕ are true. The j-collapsibility of a problem is an extremely strong computational condition, since quantified formulas with a constant number of universally quantified variables are, intuitively, very close to being CSPs. Indeed, we will combine j-collapsibility results with known CSP tractability results to obtain QCSP tractability results. In particular, we demonstrate the QCSP tractability of any constraint language having one of the following types of operations as polymorphism: semilattice operations with unit, near-unanimity operations, and Maltsev operations.

The contents of this chapter are as follows. First, we define the notions of j-collapsing and j-collapsibility; and, we give a characterization of when a j-collapsing of a quantified formula is true in terms of strategies and adversaries (Section 4.1). In particular, we show that a j-collapsing of a formula is true if and only if there is a strategy for the formula which can win against a certain restricted collection of adversaries. We then introduce a tool called *composability* for collections of adversaries to be composed, that makes it possible to show that, for certain problems $\mathsf{QCSP}(\Gamma)$, the truth of an instance formula can be inferred from the truth of its j-collapsings (Section 4.2). We then apply the developed theory to demonstrate that certain problems $\mathsf{QCSP}(\Gamma)$ are j-collapsible, and consequently tractable (Section 4.3).

4.1 Collapsings of Formulas

Note that throughout this section, we assume j to be a positive integer.

Definition 55 *A quantified formula ϕ' is a j-*collapsing *of another quantified formula ϕ if:*

- *$|Y_\phi| \leq j$ and $\phi' = \phi$, or*
- *$|Y_\phi| > j$ and there exists a subset Y' of Y_ϕ having size j such that ϕ' can be obtained from ϕ by first eliminating, from the quantifier prefix of ϕ, all variables in $Y_\phi \setminus Y'$ except for $\mathsf{first}_\phi(Y_\phi \setminus Y')$; and then replacing, in the constraint network of ϕ, all variables in $Y_\phi \setminus Y'$ with $\mathsf{first}_\phi(Y_\phi \setminus Y')$.*

*In the latter case, we say that ϕ' is the j-*collapsing of ϕ with respect to Y'.

Roughly speaking, a j-collapsing of a formula ϕ is obtained by picking j universal variables of ϕ, and then associating together the remaining universal variables.

For example, the quantified formula

$$\forall y_1 \exists x_1 \forall y_2 \forall y_3 \exists x_2 \{R_1(y_1, x_1), R_2(y_3, x_2), R_3(y_2, y_3, x_2)\}$$

has the following three 1-collapsings:

$$\forall y_1 \exists x_1 \forall y_2 \exists x_2 \{R_1(y_1, x_1), R_2(y_2, x_2), R_3(y_2, y_2, x_2)\}$$

$$\forall y_1 \exists x_1 \forall y_2 \exists x_2 \{R_1(y_1, x_1), R_2(y_1, x_2), R_3(y_2, y_1, x_2)\}$$

$$\forall y_1 \exists x_1 \forall y_3 \exists x_2 \{R_1(y_1, x_1), R_2(y_3, x_2), R_3(y_1, y_3, x_2)\}.$$

The first formula given is the j-collapsing with respect to $Y' = \{y_1\}$, obtained by associating together y_2 and y_3; the second is the j-collapsing with respect to $Y' = \{y_2\}$; and, the third is the j-collapsing with respect to $Y' = \{y_3\}$.

We have the following fact concerning the j-collapsings of a true quantified formula.

Proposition 56 *If a quantified formula ϕ is true, then for any j-collapsing ϕ' of ϕ, the quantified formula ϕ' is true.*

Interestingly, we will be able to demonstrate classes of formulas for which the converse of Proposition 56 holds–that is, classes of formulas that are *j-collapsible*.

Defi nition 57 *A class of quantified formulas is j-*collapsible *if for every quantified formula ϕ from the class, the following property holds: if all j-collapsings of ϕ are true, then ϕ is true.*

We first give an alternative characterization of the truth of a j-collapsing. Define an *adversary set* (over a set of variables Y) to be a set of functions $\tau : Y \to D$. When ϕ is a quantified formula and F is an adversary set (over Y_ϕ), we say that F is *ϕ-winnable* if there exists a strategy σ for ϕ such that for all adversaries $\tau \in F$, the assignment (σ, τ) satisfies the constraint network of ϕ; when ϕ is understandable from the context, we simply say that F is *winnable*. In this terminology, a quantified formula ϕ is true if and only if the adversary set containing all adversaries $\tau : Y_\phi \to D$ is ϕ-winnable.

When Y' is a subset of Y having size j and F is the adversary set (over Y) containing all functions $\tau : Y \to D$ having the property that $\tau(y_1) = \tau(y_2)$ for all $y_1, y_2 \in Y \setminus Y'$, we say that F is the *j-adversary set (over Y) with respect to Y'*. That is, the j-adversary set with respect to Y' contains those adversaries that set variables in Y' freely, but set all variables outside of Y' to the same value. Using the notion of j-adversary set, we can characterize the truth of a j-collapsing (of a quantified formula).

Proposition 58 *Let ϕ be a quantified formula and let Y' be a subset of Y_ϕ of size $j < |Y_\phi|$. The j-collapsing (of ϕ) with respect to Y' is true if and only if the j-adversary set (over Y_ϕ) with respect to Y' is ϕ-winnable.*

From this characterization of the truth of a particular j-collapsing, we can immediately derive the following fact, which will be useful in proving that various classes of quantified formulas are j-collapsible, for some constant j.

Proposition 59 *Let ϕ be a quantified formula and suppose j is a positive integer with $j < |Y_\phi|$. All j-collapsings (of ϕ) are true if and only if all j-adversary sets (over Y_ϕ) are ϕ-winnable.*

4.2 Composability

As mentioned, our primary objective in this chapter is to obtain QCSP tractability results by proving that restricted classes of quantified formulas are j-collapsible. The truth of any quantified formula that is j-collapsible can be decided by checking the truth of all j-collapsings of the formula. To prove j-collapsibility of a quantified formula, one needs to show that if all j-collapsings of the formula are true, then the formula itself is true. In the previous section, we showed that if the j-collapsings of a formula are true, then the formula is winnable against certain limited adversary sets called j-adversary sets. Moreover, we noted that if a formula is winnable against the set of all adversaries, then it is true. Our methodology for proving j-collapsibility is as follows. We will introduce a notion called *composability* that permits us to show–under some circumstances–that when a formula is winnable against some adversary sets, it is winnable against an adversary set that is larger than and composed from the original adversary sets. Actual j-collapsibility results are obtained by assuming the winnability of j-adversary sets, and then continually composing adversary sets known to be winnable to eventually obtain the entire set of all adversaries–and hence the truth of the formula.

We introduce the following notion of composability for adversary sets, with respect to a function μ.

Definition 60 *Let $\mu : D^k \to D$ be a function, and let $F, F_1, \ldots, F_k$ be adversary sets over the set of variables $\{y_1, \ldots, y_n\}$. We say that F* is μ-composable from $F_1, \ldots, F_k$ *if there exist sequences of partial mappings*

$$\pi^1 = \{\pi_i^1 : D^i \to D\}_{i\in[n]}, \ldots, \pi^k = \{\pi_i^k : D^i \to D\}_{i\in[n]}$$

such that for all $\tau \in F$, the following two properties hold:

- *for all $i \in [k]$, the mapping $\tau^i : \{y_1, \ldots, y_n\} \to D$ defined by*

$$\tau^i(y_j) = \pi^i_j(\tau(y_1), \ldots, \tau(y_j))$$

  for all $j \in [n]$, is total and contained in F_i, and

- *it holds that $\tau(y_j) = \mu(\tau^1(y_j), \ldots, \tau^k(y_j))$, for all $j \in [n]$.*

The key feature of this definition is the following theorem.

Theorem 61 *Let $\mu : D^k \to D$ be an operation and let the quantified formula ϕ be an instance of $\mathsf{QCSP}(\mu)$ with universal variables $y_1, \ldots, y_n$ (where for $i < j$, the variable y_i comes before y_j in the quantification order of ϕ). Suppose that F is μ-composable from $F_1, \ldots, F_k$. If ϕ is F_i-winnable for all $i \in [k]$, then ϕ is F-winnable.*

Proof: Let ϕ be a quantified formula with universal variables $y_1, \ldots, y_n$ and existential variables $x_1, \ldots, x_m$. Let $u(i)$ denote the number of universal variables coming before the ith existential variable, x_i. We define a strategy $\sigma = \{\sigma_i : D^{u(i)} \to D\}_{i \in [m]}$ where each function $\sigma_i(d_1, \ldots, d_{u(i)})$ is defined as

$$\mu(\sigma^1_i(\pi^1_1(d_1), \ldots, \pi^1_{u(i)}(d_1, \ldots, d_{u(i)})), \ldots \sigma^k_i(\pi^k_1(d_1), \ldots, \pi^k_{u(i)}(d_1, \ldots, d_{u(i)}))).$$

We show that for all $\tau \in F$, it holds that

$$(\sigma, \tau) = \mu((\sigma^1, \tau^1), \ldots, (\sigma^k, \tau^k))$$

where the τ^i are defined as in Definition 60.

For each universal variable y_i we have

$$(\sigma, \tau)(y_i) = \mu(\tau^1(y_i), \ldots, \tau^k(y_i)) = \mu((\sigma^1, \tau^1)(y_i), \ldots, (\sigma^k, \tau^k)(y_i)).$$

For each existential variable x_i we have

$$\begin{aligned}(\sigma,\tau)(x_i) &= \sigma_i(\tau(y_1),\ldots,\tau(y_{u(i)}))\\ &= \mu(\sigma_i^1(\pi_1^1(\tau(y_1)),\ldots,\pi_{u(i)}^1(\tau(y_1),\ldots,\tau(y_{u(i)}))),\\ &\qquad \ldots,\\ &\qquad \sigma_i^k(\pi_1^k(\tau(y_1)),\ldots,\pi_{u(i)}^k(\tau(y_1),\ldots,\tau(y_{u(i)}))))\\ &= \mu(\sigma_i^1(\tau^1(y_1),\ldots,\tau^1(y_{u(i)})),\ldots,\sigma_i^k(\tau^k(y_1),\ldots,\tau^k(y_{u(i)}))))\\ &= \mu((\sigma^1,\tau^1)(x_i),\ldots,(\sigma^k,\tau^k)(x_i))\end{aligned}$$

•

For convenience, we will write adversaries using tuple notation; an adversary $\tau : \{y_1,\ldots,y_n\} \to D$ will be written as the tuple $(\tau(y_1),\ldots,\tau(y_n))$. We will be concerned primarily with adversary sets that are independent in the sense that picking a value for one variable does not constrain the choices that can be made for another variable. Let us define an adversary set F (over the set of variables $\{y_1,\ldots,y_n\}$) to be an *independent adversary set* if there exist sets $D_1,\ldots,D_n \subseteq D$ such that

$$F = D_1 \times \cdots \times D_n.$$

That is, F is an independent adversary set if there exist subsets $D_1,\ldots,D_n \subseteq D$ such that F contains any and all mappings $\tau : \{y_1,\ldots,y_n\} \to D$ with $\tau(y_i) \in D_i$.

We will use the following specialized form of composability to manipulate independent adversary sets.

Definition 62 *Let $\mu : D^k \to D$ be a function, and let $F, F_1, \ldots, F_k$ be independent adversary sets, denoted by*

$$F = A_1 \times \cdots \times A_n$$

and

$$F_i = D_{i1} \times \cdots \times D_{in}$$

for sets $D_{ij} \subseteq D$ (with $i \in [k], j \in [n]$) and $A_j \subseteq D$ (with $i \in [n]$).

When $A_j \subseteq \mu(D_{1j}, \ldots, D_{kj})$ for all $j \in [n]$, we say that F is independently μ-composable *from $F_1, \ldots, F_k$; this is denoted by $F \lhd \mu(F_1, \ldots, F_k)$.*

Theorem 63 *Let $\mu : D^k \to D$ be a function, and let $F, F_1, \ldots, F_k$ be independent adversary sets such that F is independently μ-composable from $F_1, \ldots, F_k$. Then F is μ-composable from $F_1, \ldots, F_k$.*

Proof: Let us denote the independent adversary sets $F, F_1, \ldots, F_k$ as in Definition 62. For each $i \in [k]$ and $j \in [n]$, let $\rho_{ij} : A_j \to D_{ij}$ be a function such that for all $j \in [n]$ and $a \in A_j$, it holds that

$$a = \mu(\rho_{1j}(a), \ldots, \rho_{kj}(a)).$$

Define partial mappings

$$\pi^1 = \{\pi^1_i : D^i \to D\}_{i \in [n]}, \ldots, \pi^k = \{\pi^k_i : D^i \to D\}_{i \in [n]}$$

by

$$\pi^i_j(d_1, \ldots, d_j) = \rho_{ij}(d_j)$$

when $d_j \in A_j$, for all $i \in [k]$ and $j \in [n]$. It is straightforward to verify that F is μ-composable from $F_1, \ldots, F_k$ via the mappings π^i_j. •

Let $I \subseteq [n]$ be a set of indices and let $d \in D$ be a domain element. We define the adversary set $\mathcal{A}(I, d)$ as $D_1 \times \cdots \times D_n$ where $D_i = D$ if $i \in I$, and $D_i = \{d\}$ if $i \notin I$.

Proposition 64 *If all j-collapsings of a quantified formula ϕ with n universal variables are true, then for every subset $I \subseteq [n]$ of size $|I| \leq j$, and for every domain element $d \in D$, the adversary set $\mathcal{A}(I, d)$ is ϕ-winnable.*

Proof: Immediate from Proposition 59 and the fact that any subset of a winnable adversary set is also winnable. •

4.3 Applications

Using the developed machinery, we can now derive the collapsibility of a variety of constraint languages. In particular, we now give a number of results which demonstrate that certain problems of the form $\mathsf{QCSP}(\mu)$ are j-collapsible, for some j.

All of the proofs in this section have a similar structure. To show that a problem $\mathsf{QCSP}(\mu)$ is j-collapsible, we begin by assuming that all j-adversary sets are winnable, and hence that all adversary sets $\mathcal{A}(I, d)$ with $|I| \leq j$ are winnable. We then use independent μ-composability to show that more complex adversary sets are winnable, eventually showing that the adversary set containing all adversaries is winnable.

Theorem 65 *Let $\oplus : D^2 \rightarrow D$ be a semilattice operation with unit. The problem $\mathsf{QCSP}(\oplus)$ is 1-collapsible.*

Proof: Let u be the unit element of the semilattice operation $\oplus$. We assume that all adversary sets $\mathcal{A}(I, u)$ with $|I| = 1$ are ϕ-winnable, where ϕ is a quantified formula with n universal variables. We prove by induction that for each $l \in [n]$, all adversary sets $\mathcal{A}(I, u)$ with $|I| = l$ are ϕ-winnable; the base case $l = 1$ holds by assumption. Let $I \subseteq [n]$ be a set of indices of size $l + 1$, and let i, j be two distinct elements of I. We have

$$\mathcal{A}(I, u) \triangleleft \oplus(\mathcal{A}(I \setminus \{i\}, u), \mathcal{A}(I \setminus \{j\}, u))$$

as $D = \oplus(\{u\}, D) = \oplus(D, \{u\})$. •

In the tractability proof for problems $\mathsf{QCSP}(\oplus)$ given in the previous chapter (Theorem 54), it was shown that an instance ϕ of $\mathsf{QCSP}(\oplus)$ is true if and only if all formulas obtainable from ϕ by setting all $\forall$-variables but one to the unit element u are true. The proof of Theorem 65 just given yields an alternative algebraic proof of this fact.

Theorem 66 *Let $\mu : D^k \rightarrow D$ be a near-unanimity operation of rank $k \geq 3$. The problem* $\mathsf{QCSP}(\mu)$ *is $(k-1)$-collapsible.*

Proof: Fix $d \in D$ to be any domain element. We assume that all adversary sets $\mathcal{A}(I, d)$ with $|I| = k - 1$ are ϕ-winnable, where ϕ is a quantified formula with n universal variables. We prove by induction that for each l with $(k-1) \leq l \leq n$, all adversary sets $\mathcal{A}(I, d)$ with $|I| = l$ are ϕ-winnable; the base case $l = k - 1$ holds by assumption. Let $I \subseteq [n]$ be a set of indices of size $l + 1$, and let $i_1, \ldots, i_k$ be distinct elements of I. We have

$$\mathcal{A}(I, d) \triangleleft \mu(\mathcal{A}(I \setminus \{i_1\}, d), \ldots, \mathcal{A}(I \setminus \{i_k\}, d))$$

as $D = \mu(\{d\}, D, \ldots, D) = \cdots = \mu(D, \ldots, D, \{d\})$. •

The *dual discriminator operation* is defined to be the near-unanimity operation $\delta : D^3 \rightarrow D$ of rank 3 such that $\delta(x, y, z) = z$ when no two of x, y, z are equal. By the previous theorem, every problem of the form $\mathsf{QCSP}(\delta)$ is 2-collapsible. We can show that when δ is the dual discriminator operation, the problem $\mathsf{QCSP}(\delta)$ is in fact 1-collapsible.

Theorem 67 *Let $\delta : D^k \rightarrow D$ be the dual discriminator operation on D. The problem* $\mathsf{QCSP}(\delta)$ *is 1-collapsible.*

Proof: Fix $a, b \in D$ to be distinct domain elements. We assume that all adversary sets $\mathcal{A}(I, d)$ with $|I| = 1$ and $d \in \{a, b\}$ are ϕ-winnable, where ϕ is a quantified formula with n universal variables. We prove by induction that for each $l \in [n]$, all adversary sets $\mathcal{A}(I, d)$ with $|I| = l$ and $d \in \{a, b\}$ are ϕ-winnable; the base case $l = 1$ holds by assumption. Let $I \subseteq [n]$ be a set of indices of size $l+1$, and let i, j be distinct elements of I. We have

$$\mathcal{A}(I, a) \lhd \delta(\mathcal{A}(I \setminus \{i\}, a), \mathcal{A}(I \setminus \{i\}, b), \mathcal{A}(I \setminus \{j\}, a))$$

as $D = \delta(\{a\}, \{b\}, D) = \delta(D, D, \{a\})$ and $a = \delta(a, b, a)$. Similarly, we have

$$\mathcal{A}(I, b) \lhd \delta(\mathcal{A}(I \setminus \{i\}, b), \mathcal{A}(I \setminus \{i\}, a), \mathcal{A}(I \setminus \{j\}, b)).$$

•

A Maltsev operation $m : D^3 \rightarrow D$ is a ternary operation satisfying the identities $m(x, x, y) = m(y, x, x) = y$.

Theorem 68 *Let $m : D^3 \rightarrow D$ be a Maltsev operation. The problem $\mathsf{QCSP}(d)$ is 1-collapsible.*

Proof: Fix $d \in D$ to be any domain element. We assume that all adversary sets $\mathcal{A}(I, d)$ with $|I| = 1$ are ϕ-winnable, where ϕ is a quantified formula with n universal variables. We prove by induction that for each $l \in [n]$, all adversary sets $\mathcal{A}(I, d)$ with $|I| = l$ are ϕ-winnable; the base case $l = 1$ holds by assumption. Let $I \subseteq [n]$ be a set of indices of size $l + 1$, and let i, j be distinct elements of I. We have

$$\mathcal{A}(I, d) \lhd m(\mathcal{A}(I \setminus \{i\}, d), \mathcal{A}(I \setminus \{i\}, d), \mathcal{A}(I \setminus \{j\}, d))$$

as $D = m(\{d\}, \{d\}, D) = m(D, D, \{d\})$. •

We have just shown that certain problems $\mathsf{QCSP}(\mu)$ are j-collapsible. From these results, we can infer the tractability of the problems $\mathsf{QCSP}(\mu)$.

Theorem 69 *Suppose that μ is a semilattice operation with unit, a near-unanimity operation, or a Maltsev operation. Then, the problem $\mathsf{QCSP}(\mu)$ is tractable.*

Proof: Let μ be an operation of one of the described types. By previous results (see Theorems 34, 35, and 36), the problem $\mathsf{CSP}(\mu)$ is tractable. By Theorems 65, 66, and 68, the problem $\mathsf{QCSP}(\mu)$ is j-collapsible for some constant j. An algorithm for $\mathsf{QCSP}(\mu)$ is the following: for all j-collapsings ϕ' of $\mathsf{QCSP}(\mu)$, check the truth of ϕ' by converting ϕ' to an instance of $\mathsf{CSP}(\mu)$ (as in Theorem 69) and using a polynomial-time algorithm for $\mathsf{CSP}(\mu)$. Output true if and only if all j-collapsings ϕ' are true. Note that, when j is a constant, there are polynomially many j-collapsings of a quantified formula, and all of them can be computed in polynomial time. •

Chapter 5

Complexity of 2-Semilattice Polymorphisms

In this chapter, we investigate constraint languages that have a *2-semilattice operation* as polymorphism. A 2-semilattice operation is a binary operation $\star$ that satisfies the semilattice identities restricted to two variables, namely, the identities $x \star x = x$ (idempotence), $x \star y = y \star x$ (commutativity), and $(x \star x) \star y = x \star (x \star y)$ (restricted associativity). 2-semilattices constitute a natural generalization of semilattices; semilattices were one of the initial classes of polymorphisms shown to guarantee CSP tractability [46], and since then 2-semilattices have been shown to imply CSP tractability by Bulatov [6]. We prove a full classification of 2-semilattice polymorphisms for QCSPs, showing that some such polymorphisms guarantee QCSP tractability, while others do not.

We would like to highlight three reasons as to why we believe our study of 2-semilattice polymorphisms in the QCSP setting is interesting.

First, as pointed out previously [6], 2-semilattice polymorphisms play an important role in the investigation of *maximal constraint languages,* which are constraint languages that can express any relation when augmented with any relation not expressible by the language. Because a constraint language that can express all relations is intractable, maximal constraint languages are the largest constraint languages that could possibly be tractable (in either the CSP or QCSP setting); hence, studying maximal constraint languages allows one to obtain the most general tractability results possible. (It is worth noting here that all of the tractability results identified by Schaefer's theorem apply to maximal constraint languages.) It follows from a theorem of Rosenberg [50] that maximal constraint languages can be classified into five types; one of the types consists of constraint languages having a non-projection binary idempotent operation as polymorphism. Because 2-semilattice operations are indeed operations of this type, the present work constitutes a step towards understanding this type. We mention that in the CSP setting, the tractability of 2-semilattices has been leveraged to give complete complexity classifications of maximal constraint languages for domains of size three and four [15, 6].

Second, our tractability proofs make use of and validate the collapsibility machinery for proving QCSP tractability that we have developed the previous chapter. In Chapter 4, it was demonstrated that for a number of types of polymorphisms μ where $\mathsf{CSP}(\mu)$ is tractable, the problem $\mathsf{QCSP}(\mu)$ is j-collapsible, and hence also tractable. We give another class of constraint languages having this property, by showing that when $\star$ is a 2-semilattice operation such that $\mathsf{QCSP}(\star)$ is tractable, the problem $\mathsf{QCSP}(\star)$ is 1-collapsible. This provides further evidence that collapsibility is a fruitful and useful tool for studying QCSP complexity.

Third, although all 2-semilattice polymorphisms are tractable in the CSP setting, 2-semilattice polymorphisms intriguingly yield two modes of behavior in the QCSP setting: some 2-semilattice polymorphisms guarantee QCSP tractability, while others (provably) do not. This is surprising in light of the fact that for all other types of polymorphisms of non-trivial constraint languages that have been investigated, polymorphisms that guarantee CSP tractability also guarantee QCSP tractability. In fact, our results imply the first and only known examples of non-trivial constraint languages that are CSP tractable, but QCSP intractable.[1] The existence of such constraint languages implies that the boundary between tractability and intractability in the QCSP context is genuinely different from the corresponding boundary in the CSP context.

The contents of this chapter are as follows. We begin by stating our classification theorem on 2-semilattices and we derive some consequences of this theorem (Section 5.1). We then give the tractability and intractability proofs for the classification theorem (Sections 5.2 and 5.3).

5.1 Classification of 2-semilattices

This chapter presents a complete complexity classification of 2-semilattice operations in quantified constraint satisfaction. In this section, we formally state the classification theorem (Theorem 70) and discuss some of its implications; then, we prove the theorem in two parts.

Every 2-semilattice operation $\star : D^2 \to D$ induces a directed graph $\mathcal{G}^\star = (D, E)$ with edge set $E = \{(a,b) \in D \times D : a \star b = b\}$. We use $\mathcal{C}^\star$ to denote the set of strongly connected components (or *components,* for short) of $\mathcal{G}^\star$, and let $\leq$ be the binary relation on $\mathcal{C}^\star$ where for $C_1, C_2 \in \mathcal{C}^\star$, it holds that $C_1 \leq C_2$ if and only if there exist vertices $v_1 \in C_1$, $v_2 \in C_2$ such that there is a path in $\mathcal{G}^\star$ from v_1 to v_2. It is straightforward to verify that $\leq$ is a partial order. We say that $C \in \mathcal{C}^\star$ is a *minimal component* if it is

[1]Here, by a non-trivial constraint language, we mean a constraint language that includes each constant as a relation. When μ is an idempotent operation, $\mathsf{Inv}(\mu)$ contains each constant as a relation and hence the problem $\mathsf{QCSP}(\mu)$ may be viewed as being over a non-trivial constraint language.

minimal with respect to $\leq$, that is, for all $C' \in \mathcal{C}^\star$, $C' \leq C$ implies $C' = C$.

Our classification theorem demonstrates that 2-semilattice operations give rise to two modes of behavior in QCSP complexity, depending on the structure of the graph $\mathcal{G}^\star$.

Theorem 70 *Let $\star : D^2 \to D$ be a 2-semilattice operation. If there is a unique minimal component in $\mathcal{C}^\star$, then $\mathsf{QCSP}(\star)$ is decidable in polynomial time. Otherwise, $\mathsf{QCSP}(\star)$ is coNP-hard.*

Proof: Proved as Theorems 73 and 75 below. ●

One implication of this classification theorem is a complete classification of semilattice operations in QCSP complexity. In Chapters 3 and 4, we proved the QCSP tractability of semilattice operations with unit. Our classification of 2-semilattices allows us to derive the intractability of all other semilattice operations.

Corollary 71 *Let $\star : D^2 \to D$ be a semilattice operation. If $\star$ has a unit element, then $\mathsf{QCSP}(\star)$ is decidable in polynomial time. Otherwise, $\mathsf{QCSP}(\star)$ is coNP-hard.*

Proof: When $\star$ is a semilattice operation, it is straightforward to verify that each component in $\mathcal{C}^\star$ is of size one. Hence, $\mathcal{C}^\star$ has a unique minimal component if and only if $\star$ has a unit element. ●

Another implication of our classification theorem is the tractability of all *commutative conservative operations*. A commutative conservative operation is a binary operation $\star : D^2 \to D$ that is commutative and conservative; we say that a binary operation $\star$ is conservative if for all $x, y \in D$ it holds that $x \star y \in \{x, y\}$. Such operations were studied in the context of the CSP by Bulatov and Jeavons [11], prior to Bulatov's study of 2-semilattices [6].

Corollary 72 *Let $\star : D^2 \to D$ be a commutative conservative operation. The problem* $\mathsf{QCSP}(\star)$ *is decidable in polynomial time.*

Proof: It is straightforward to verify that when $\star$ is a commutative conservative operation, the relation $\leq$ on $\mathcal{C}^\star$ is a total ordering, and hence has a unique minimal component. •

5.2 Tractability

We give two different proofs of the tractability of 2-semilattice operations $\star$ identified to be tractable by our classification theorem. We first give a relatively simple and short proof.

Theorem 73 *Let $\star : D^2 \to D$ be a 2-semilattice operation. If there is a unique minimal component in $\mathcal{C}^\star$, then* $\mathsf{QCSP}(\star)$ *is decidable in polynomial time.*

Proof: Let $\mu : D^{2^k} \to D$ be the function defined by

$$\mu(x_1, \ldots, x_{2^k}) = (((x_1 \star x_2) \star (x_3 \star x_4)) \star \cdots)$$

where the right hand side is a balanced binary tree of depth k with leaves $x_1, \ldots, x_{2^k}$. Since any relation having $\star$ as polymorphism also has μ as polymorphism, every instance of $\mathsf{QCSP}(\star)$ is an instance of $\mathsf{QCSP}(\mu)$, and it suffices to show that $\mathsf{QCSP}(\mu)$ is polynomial-time decidable.

We show that the problem $\mathsf{QCSP}(\mu)$ is $(2^k - 1)$-collapsible; the result then follows by the argument given in the proof of Theorem 69. Our proof of collapsibility uses the notation and follows the structure of the collapsibility proofs in Section 4.3.

Fix an element b in the minimal component of $\mathcal{C}^\star$. For every element $d \in D$, there exists a path from b to d in $\mathcal{G}^\star$; this is because for every component C of $\mathcal{C}^\star$ it holds that

$B \leq C$, where B denotes the minimal component of $\mathcal{C}^\star$. Let k be a sufficiently large integer such that for every $d \in D$, there is a path from b to d of length less than or equal to k.

We assume that all adversary sets $\mathcal{A}(I, b)$ with $|I| = 2^k - 1$ are ϕ-winnable, where ϕ is a quantified formula with n universal variables. We prove by induction that for each l with $(2^k - 1) \leq l \leq n$, all adversary sets $\mathcal{A}(I, b)$ with $|I| = l$ are ϕ-winnable; the base case $l = 2^k - 1$ holds by assumption. Let $I \subseteq [n]$ be a set of indices of size $l + 1$, and let $i_1, \ldots, i_{2^k}$ be distinct elements of I. We claim that

$$\mathcal{A}(I, b) \lhd \mu(\mathcal{A}(I \setminus \{i_1\}, b), \ldots, \mathcal{A}(I \setminus \{i_{2^k}\}, b)).$$

Looking at coordinate positions $i_1, \ldots, i_{2^k}$, it is necessary to show surjectivity of the function $\mu(x_1, \ldots, x_{2^k})$ where one of the arguments has been instantiated with b. By symmetry, it suffices to show surjectivity of the function $\mu(b, x_2, \ldots, x_{2^k})$. Identifying the variables $x_{2^i+1}, \ldots, x_{2^{i+1}}$ to be equivalent for each $i = 0, \ldots, k-1$, it suffices to show that the function

$$(\cdots((b \star x'_0) \star x'_1) \star \cdots \star x'_{k-1})$$

is surjective. This follows immediately from our choice of k: for any element $d \in D$, there is a path from b to d in $\mathcal{G}^\star$ of length k. Note that every vertex in $\mathcal{G}^\star$ has a self-loop, so as long as there is a path from b to d with length less than or equal to k, there is a path from b to d with length equal to k. •

We have just shown that for certain 2-semilattice operations $\star$, the problem $\mathsf{QCSP}(\star)$ is polynomial-time tractable. For these operations, it was proved that $\mathsf{QCSP}(\star)$ is j-collapsible for *some* constant j. However, the proof that was just given demonstrated j-collapsibility for constants j that could be arbitrarily large, depending on the operation $\star$. We now refine this result by proving the strongest possible statement concerning the collapsibility of $\mathsf{QCSP}(\star)$: the problem $\mathsf{QCSP}(\star)$ is 1-collapsible whenever $\mathsf{QCSP}(\star)$ is tractable.

Theorem 74 *Let $\star : D^2 \to D$ be a 2-semilattice operation. If there is a unique minimal component in $\mathcal{C}^\star$, then $\mathsf{QCSP}(\star)$ is 1-collapsible.*

Proof: Fix d_0 to be any element of the unique minimal component of $\mathcal{C}^\star$. For every element $d \in D$, there exists a path from d_0 to d in $\mathcal{G}^\star$. Hence, it is possible to select sufficiently large integers K and L and elements $\{d_i^j\}_{i\in[L],j\in[K]}$ so that:

- each of the sets
$$P^1 = \{d_0^1, d_1^1, \ldots, d_L^1\}$$
$$\vdots$$
$$P^K = \{d_0^K, d_1^K, \ldots, d_L^K\}$$
is a path in the sense that, for all $j \in [K]$, all of the pairs
$$(d_0^j, d_1^j), (d_1^j, d_2^j), \ldots, (d_{L-1}^j, d_L^j)$$
are edges in $\mathcal{G}^\star$; and,

- all elements of D lie on one of these paths, that is, $D = \cup_{j=1}^K P^j$.

Here, we use $d_0^1, \ldots, d_0^K$ as alternative notation for d_0, that is,
$$d_0 = d_0^1 = \cdots = d_0^K.$$

For $i = 0, \ldots, L$ we define
$$D_i \stackrel{\text{def}}{=} \{d_i^1, \ldots, d_i^k\},$$

and we define
$$E_i \stackrel{\text{def}}{=} D_0 \cup \cdots \cup D_i.$$

Notice that $E_L = D$.

We proceed with the proof of 1-collapsibility by fixing a problem instance and assuming that every 1-adversary set is winnable. In particular, we assume that each of the sets

$$D \times \{d_0\} \times \cdots \times \{d_0\}$$

$$\vdots$$

$$\{d_0\} \times \cdots \times \{d_0\} \times D$$

is winnable. Our goal is to prove that $D \times \cdots \times D$ is winnable.

We prove by induction that for all $i = 0, \ldots, L$ the set $E_i \times \cdots \times E_i$ is winnable. The base case $i = 0$ holds by assumption. For the induction, assume that $E_i \times \cdots \times E_i$ is winnable (where $i \geq 0$); we show that $E_{i+1} \times \cdots \times E_{i+1}$ is winnable.

We first show that $E_{i+1} \times E_i \times \cdots \times E_i$ is winnable. We have

$$(E_{i+1} \times E_0 \times \cdots \times E_0) \lhd (E_i \times E_i \times \cdots \times E_i) \star (D \times \{d_0\} \times \cdots \times \{d_0\}).$$

Then, we have

$$(E_{i+1} \times E_1 \times \cdots \times E_1) \lhd (E_i \times E_i \times \cdots \times E_i) \star (E_{i+1} \times E_0 \times \cdots \times E_0)$$

and

$$(E_{i+1} \times E_2 \times \cdots \times E_2) \lhd (E_i \times E_i \times \cdots \times E_i) \star (E_{i+1} \times E_1 \times \cdots \times E_1).$$

Continuing in this manner, we can show that $E_{i+1} \times E_i \times \cdots \times E_i$ is winnable. By symmetric arguments, we can show the winnability of the sets

$$(E_i \times E_{i+1} \times E_i \times \cdots \times E_i)$$

$$\vdots$$

$$(E_i \times E_i \times \cdots \times E_i \times E_{i+1}).$$

We have the winnability of $E_{i+1} \times E_{i+1} \times E_i \times \cdots E_i$ by

$$(E_{i+1} \times E_{i+1} \times E_i \times \cdots E_i) \triangleleft (E_{i+1} \times E_i \times E_i \times \cdots \times E_i) \star (E_i \times E_{i+1} \times E_i \times \cdots \times E_i)$$

and we have the winnability of $E_{i+1}^3 \times E_i \times \cdots \times E_i$ by

$$(E_{i+1}^3 \times E_i \times \cdots \times E_i) \triangleleft (E_{i+1}^2 \times E_i \times E_i \times \cdots \times E_i) \star (E_i^2 \times E_{i+1} \times E_i \times \cdots \times E_i).$$

Proceeding in this fashion, we have the winnability of $E_{i+1} \times \cdots \times E_{i+1}$, as desired. •

5.3 Intractability

We complete the classification of 2-semilattices by proving a complement to Theorems 73 and 74, namely, that the remaining 2-semilattices give rise to QCSPs that are intractable.

Theorem 75 *Let $\star : D^2 \to D$ be a 2-semilattice operation. If there are two or more minimal components in $\mathcal{C}^\star$, then $\mathsf{QCSP}(\star)$ is coNP-hard (even when restricted to $\forall\exists$-formulas).*

We remark that Theorem 75 is a *local* intractability result: for 2-semilattice operations $\star$ satisfying the hypothesis of the theorem, we will prove that there is a finite subset Γ of $\mathsf{Inv}(\{\star\})$ such that $\mathsf{QCSP}(\Gamma)$ is coNP-hard.

Proof: We show coNP-hardness by reducing from the propositional tautology problem. Let $C(y_1, \ldots, y_n)$ be an instance of this problem, where C is a circuit with input gates having labels $y_1, \ldots, y_n$. We assume without loss of generality that all non-input gates of C are either AND or NOT gates, and assign all non-input gates labels $x_1, \ldots, x_m$. The quantifier prefix of the resulting quantified formula is $\forall y_1 \ldots \forall y_n \exists x_1 \ldots \exists x_m$. Let

B_0 and B_1 be distinct minimal components in $\mathcal{C}^\star$, and let b_0 and b_1 be elements of B_0 and B_1, respectively. The constraint network of the resulting quantified formula is constructed as follows.

For each AND gate x_i with inputs $v, v' \in \{y_1, \ldots, y_n\} \cup \{x_1, \ldots, x_m\}$, include the four constraints:

- $(v \in B_1) \wedge (v' \in B_1) \Rightarrow (x_i = b_1)$
- $(v \in B_0) \wedge (v' \in B_1) \Rightarrow (x_i = b_0)$
- $(v \in B_1) \wedge (v' \in B_0) \Rightarrow (x_i = b_0)$
- $(v \in B_0) \wedge (v' \in B_0) \Rightarrow (x_i = b_0)$

For each NOT gate x_i with input $v \in \{y_1, \ldots, y_n\} \cup \{x_1, \ldots, x_m\}$, include the two constraints:

- $(v \in B_0) \Rightarrow (x_i = b_1)$
- $(v \in B_1) \Rightarrow (x_i = b_0)$

For the output gate x_o, include the constraint:

- $(x_o \in B_0) \Rightarrow$ FALSE

It is fairly straightforward to verify that each of the given constraints has the $\star$ operation as polymorphism; the key fact is that (for $i \in \{0, 1\}$) multiplying any element of D by an element c outside of B_i yields an element c' outside of B_i. (If not, there is an edge from c to c' by restricted associativity of $\star$; this edge gives a contradiction to the minimality of B_i).

We verify the reduction to be correct as follows.

Suppose that the original circuit was a tautology. Let $f : \{y_1, \ldots, y_n\} \to D$ be any assignment to the $\forall$-variables of the quantified formula. Define $f' : \{y_1, \ldots, y_n\} \to (B_0 \cup B_1)$ by $f'(y_i) = f(y_i)$ if $f(y_i) \in B_0 \cup B_1$, and as an arbitrary element of $B_0 \cup B_1$ otherwise. The AND and NOT gate constraints force each x_i to have either the value b_0 or b_1 under f'; it can be verified that the assignment taking x_i to its forced value and y_i to $f(y_i)$ satisfies all of the constraints.

Suppose that the original circuit was not a tautology. Let $g : \{y_1, \ldots, y_n\} \to \{0, 1\}$ be an assignment making the circuit C false. Let $g' : \{y_1, \ldots, y_n\} \to D$ be an assignment to the $\forall$-variables of the quantified formula such that $g'(y_i) \in B_{g(y_i)}$ for all i. Under the assignment g', the only assignment to the $\exists$-variables x_i satisfying the AND and NOT gate constraints is the mapping taking x_i to b_0 if the gate with label x_i has value 0 under g, and b_1 if the gate with label x_i has value 1 under g. Hence, if all of these constraints are satisfied, then the output gate constraint must be falsified. We conclude that no assignment to the $\exists$-variables x_i satisfies all of the constraints under the assignment g', and so the quantified formula is false. ●

Chapter 6

Conclusions

Projection 50 years ahead is difficult, so I have eased my problem by offering problems and projections for only 49 years. My projections concern areas in which I have worked . . . However, I am surely not up-to-date in any of these areas, so some of what I project for the future may have already happened.

– John McCarthy, Journal of the ACM 50th Anniversary Issue

In this dissertation, we have studied restricted versions of the quantified constraint satisfaction problem where all relations must come from a constraint language–that is, problems of the form $\mathsf{QCSP}(\Gamma)$ for a constraint language Γ. Our contributions included the introduction and application of a new technique, *collapsibility*, for proving polynomial-time tractability results for $\mathsf{QCSP}(\Gamma)$ problems. We believe that the work presented in this dissertation serves as significant evidence of the fruitfulness of the polymorphism-based algebraic approach to studying the constraint satisfaction problem and variants thereof.

The broad research program that we have contributed to is that of classifying, for all constraint languages Γ over a finite domain, the complexity of $\mathsf{QCSP}(\Gamma)$. We certainly look forward to future contributions to this program, and conclude by indicating three directions of research that may prove to be interesting.

Non-idempotent operations. Our investigation was solely concerned with problems of the form $\mathsf{QCSP}(\mu)$ for μ an *idempotent* operation. It appears that different techniques may be necessary to study problems of the form $\mathsf{QCSP}(f)$ where f is a *non-idempotent* operation, or more generally, problems of the form $\mathsf{QCSP}(F)$ where F is a set of operations containing non-idempotent operations. We remark that in the case of the CSP, there is a fairly direct reduction from any problem $\mathsf{CSP}(F)$ where F is an arbitrary set of operations, to a problem of the form $\mathsf{CSP}(F')$ where F' is a set of idempotent operations [14]. Such a reduction does not seem to exist in the case of the QCSP.

Relatively quantified formulas. In this dissertation, we studied quantified formulas where universal and existential quantification was over the entire domain D of the formula. It may be of interest in further investigations to consider the role of *relative quantification,* where quantification over (perhaps selected) subsets of the domain D is permitted. In particular, studying the behavior of relative universal quantification in the $\mathsf{QCSP}(\Gamma)$ framework may result in novel theory. In a problem of the form $\mathsf{QCSP}(\Gamma)$, relative existential quantification over a subset D' of the domain D can be "simulated" by adding D' (viewed as an arity 1 relation) to the constraint language Γ.

It is fairly straightforward to show, using collapsibility proofs, that near-unanimity polymorphisms and Maltsev polymorphisms are still tractable even if relatively quantification over arbitrary subsets of the domain is permitted. The situation is different for semilattice polymorphisms (and 2-semilattice polymorphisms). Suppose that

$\oplus : D^2 \to D$ is a semilattice operation with unit u and the restriction of $\oplus$ to $D \setminus \{u\}$ is a semilattice operation without unit. We have shown that $\mathsf{QCSP}(\oplus)$, in the absence of relative quantification, is polynomial-time tractable; however, if we modify the problem $\mathsf{QCSP}(\oplus)$ by permitting relative universal quantification over the domain $D \setminus \{u\}$, we obtain a class of QCSP problems that is coNP-hard; this can be proved using ideas similar to those in the proof of Theorem 75.

The exact complexity of 2-semilattices. We have shown that 2-semilattice operations $\star$ yield two modes of behavior in QCSP complexity: a problem of the form $\mathsf{QCSP}(\star)$ is either polynomial-time tractable or coNP-hard. A natural question raised by this dichotomy result is that of determining the exact complexity of the problems $\mathsf{QCSP}(\star)$ shown to be coNP-hard. We conjecture that such problems $\mathsf{QCSP}(\star)$ are contained in coNP, and hence coNP-complete. We expect investigation of this question to lead to deep and beautiful theory.

Bibliography

[1] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[2] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. Texts in Theoretical Computer Science – An EATCS series. Springer-Verlag, Berlin, 2nd edition, 1995.

[3] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with boolean blocks, part I: Post's lattice with applications to complexity theory. *ACM SIGACT-Newsletter*, 34(4):38–52, 2003.

[4] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with boolean blocks, part II: constraint satisfaction problems. *ACM SIGACT-Newsletter*, 35(1):22–35, 2004.

[5] F. Börner, A. Bulatov, A. Krokhin, and P. Jeavons. Quantified constraints: Algorithms and complexity. In *Computer Science Logic 2003*, 2003.

[6] Andrei Bulatov. Combinatorial problems raised from 2-semilattices. Manuscript.

[7] Andrei Bulatov. A dichotomy theorem for constraints on a three-element set. In *Proceedings of 43rd IEEE Symposium on Foundations of Computer Science*, pages 649–658, 2002.

[8] Andrei Bulatov. Malt'sev constraints are tractable. Technical Report PRG-RR-02-05, Oxford University, 2002.

[9] Andrei Bulatov. Tractable conservative constraint satisfaction problems. In *Proceedings of 18th IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 321–330, 2003.

[10] Andrei Bulatov and Victor Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *FOCS'03*, pages 562–572, 2003.

[11] Andrei Bulatov and Peter Jeavons. Tractable constraints closed under a binary operation. Technical Report PRG-TR-12-00, Oxford University, 2000.

[12] Andrei Bulatov and Peter Jeavons. Algebraic structures in combinatorial problems. Technical Report MATH-AL-4-2001, Technische Universitat Dresden, 2001.

[13] Andrei Bulatov and Peter Jeavons. An algebraic approach to multi-sorted constraints. In *CP 2003*, 2003.

[14] Andrei Bulatov, Andrei Krokhin, and Peter Jeavons. Constraint satisfaction problems and finite algebras. In *Proceedings 27th International Colloquium on Automata, Languages, and Programming – ICALP'00*, volume 1853 of *Lecture Notes In Computer Science*, pages 272–282, 2000.

[15] Andrei Bulatov, Andrei Krokhin, and Peter Jeavons. The complexity of maximal constraint languages. In *ACM Symposium on Theory of Computing*, pages 667–674, 2001.

[16] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Information and Computation*, 117(1):12–18, 1995.

[17] Hubie Chen and Stephen Chong. Owned policies for information security. In *17th IEEE Computer Security Foundations Workshop (CSFW)*, 2004.

[18] Hubie Chen and Victor Dalmau. (Smart) look-ahead arc consistency and the pursuit of CSP tractability. In *Principles and Practice of Constraint Programming - CP 2004*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[19] Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(1):1–12, 1996.

[20] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classification of Boolean Constraint Satisfaction Problems*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2001.

[21] Victor Dalmau. Some dichotomy theorems on constant-free quantified boolean formulas. Technical Report LSI-97-43-R, Llenguatges i Sistemes Informàtics - Universitat Politècnica de Catalunya, 1997.

[22] Victor Dalmau. A new tractable class of constraint satisfaction problems. In *6th International Symposium on Artificial Intelligence and Mathematics*, 2000.

[23] Victor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Constraint Programming '02*, LNCS, 2002.

[24] Victor Dalmau and Justin Pearson. Closure functions and width 1 problems. In *CP 1999*, pages 159–173, 1999.

[25] R. Dechter and A. Dechter. Structure driven algorithms for truth maintenance. *Artificial Intelligence Journal*, 82:1–20, 1996.

[26] William Dowling and Jean Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[27] 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(1):57–104, 1998.

[28] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph heomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[29] Ian Gent and Andrew Rowley. Encoding connect-4 using quantified boolean formulae. Technical Report APES-68-2003, 2003.

[30] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural csp decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.

[31] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. In *FOCS 2003*, pages 552–561, 2003.

[32] P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Ser.B*, 48:92–110, 1990.

[33] A. Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16:14–21, 1951.

[34] Peter Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.

[35] Peter Jeavons, David Cohen, and Martin Cooper. Constraints, consistency, and closure. *Articial Intelligence*, 101(1-2):251–265, 1998.

[36] Marek Karpinski, Hans Kleine Büning, and Peter H. Schmitt. On the computational complexity of quantified horn clauses. In *CSL 1987*, pages 129–137, 1987.

[37] D. Kavvadias and M. Sideri. The inverse satisfiability problem. *SIAM Journal on Computing*, 28(1):152–163, 1998.

[38] Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2001.

[39] L. M. Kirousis and P. G. Kolaitis. The complexity of minimal satisfiability problems. In *Proceedings 18th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2010 of *Lecture Notes in Computer Science*, pages 407–418. Springer-Verlag, 2001.

[40] Ph.G. Kolaitis and M.Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61:302–332, 2000.

[41] Phokion Kolaitis. Constraint satisfaction, databases, and logic. In *Proceedings of IJCAI 2003,* pages 1587–1595, 2003.

[42] A. Krokhin, A. Bulatov, and P. Jeavons. The complexity of constraint satisfaction: An algebraic approach.

[43] M. Mneimneh and K. Sakallah. Computing vertex eccentricity in exponentially large graphs: Qbf formulation and solution. In *Sixth International Conference on Theory and Applications of Satisfiability Testing,* 2003.

[44] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[45] C. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[46] P.G.Jeavons, D.A.Cohen, and M.Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.

[47] R. Pöschel and L. A. Kalužnin. *Funktionen- and Relationenalgebren*. Birkhäuser, 1979.

[48] L. Purvis and P. Jeavons. Constraint tractability theory and its application to the product development process for a constraint-based scheduler. In *Proceedings of 1st International Conference on The Practical Application of Constraint Technologies and Logic Programming - PACLP'99*, pages 63–79. Practical Applications Company, 1999.

[49] J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[50] I.G. Rosenberg. Minimal clones I: the five types. In *Lectures in Universal Algebra (Proc. Conf. Szeged 1983)*, volume 43 of *Colloq. Math. Soc. Janos Bolyai*, pages 405–427. North-Holland, 1986.

[51] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978.

[52] U. Schoning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.

[53] E. Schwalb and L. Vila. Temporal constraints: a survey. *Constraints*, 3(2–3):129–149, 1998.

[54] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.