Constraint Satisfaction Problems in the Logic LFP+Rank

by

Aklilu Habte

Bachelor of Science, Ryerson University, 2013

Bachelor of Engineering, Ryerson University, 1998

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Applied Mathematics

Toronto, Ontario, Canada, 2015

Author's Declaration

Abstract

Constraint Satisfaction Problems in the Logic LFP+Rank

Master of Science 2015

Aklilu Habte

Applied Mathematics

Ryerson University

Constraint satisfaction problems (CSPs) are one of the central topics in theoretical computer science, in particular, in the area of artificial intelligence. Their computational complexity is due to relatively recent results from areas of mathematics, including finite-model-theory, algebra and graph homomorphisms.

The main conjecture by Feder and Vardi states that any CSP over a finite relational template is either in **P** or is **NP**-complete. Further, it amounts to showing that every non **NP**-complete CSP can be expressed as an extension of *first-order* logic.

A finite template is *Mal'tsev*, a compatible algebraic operation, which is closely related to an affine space over a finite field. The so-called *Bulatov-Dalmau* algorithm, a natural generalization of the *Gaussian* elimination on vector spaces, shows such CSPs are tractable. In this work, we prove that CSPs described over a finite template Mal'tsev are expressible in logic LFP+rnk, providing a logical proof that such CSPs are tractable.

Acknowledgements

Over the years, Ryerson University has been a second home for me. Thank you Ryerson for offering me valuable career paths. In particular, I would like to thank the departments of Engineering, Computer Science and Mathematics.

My deepest thanks go to my supervisor Dr. Dejan Delić. I have never met such a person who is so organized, skill-full and yet very knowledgeable about the field. I have learned so much that can not be found in any text books or class rooms. Thank you Dr. Delić.

In completing my graduate studies, the faculty and staff members of the Department have been of great support. I would like to thank Dr. Anthony Bonato, Dr. Jean-Paul Pascal, Dr. Katrin Rohlf, Dr. Pawel Pralat, Dr. Pablo Olivares, Dr. Sebastian Ferrando and Dr. Silvana Illie for shaping my knowledge.

I would like to thank Steve Kanellis, department IT specialist, for trouble-shooting all computer related issues. Also, I would like to thank Kathy Peter, Luisa Chan and Teresa Lee for their relentless service.

From the Department of Computer Science, I would like to thank Dr. Mikhail Soutchanski for shaping my interest on the field early on; and Dr. Jelena Misic & Dr. Vojislav Misic for their support.

Thank you fellow graduates for being part family, part friend.

In closing, I would like to thank my entire family for their unconditional love and support. You are always my heroes.

Dedication

To

my mother *Lul Amare*
my father *Hailemichael Habte*
and my sister *Tsehay Habte.*

Contents

Chapter 1

Introduction

In general, the constraint satisfaction problem (CSP) is **NP**-complete. The motivating problem is to find under what circumstances one can design a polynomial time algorithm for solving it. Tractable cases can be achieved by restricting the relation that appear in a constraint to belong to a given set Γ. Identifying all sets Γ of relations over a finite domain leads to a class instance of CSP(Γ) that guarantees tractability [2, 3]. Hence, we focus on restricted CSP in which an algorithm decides whether there is a solution or not in polynomial time. Restricted CSPs that are now widely studied, includes `2-SATISFIABILITY`, `GRAPH H-COLOURABILITY`. Constraint satisfaction problems cover a wide range of areas of study. It arises in the areas of algebra, combinatorics, artificial intelligence (AI), graph theory, logic and other domains. For further discussion on various approaches to the study of constraint satisfaction problems, see [7].

In Chapter 4, we provide an original work, a logical proof to the so called Bulatov-Dalmau algorithm, that decides whether an instance of CSP has a solution or not in polynomial time. As a result, we show an implementation of the algorithm on finite CSP, via *least fixed-point*

(LFP) logic with matrix rank (*rnk*) operator.

The details of LFP, an extension of *first-order* (FO) logic, and the expressive power of the rank of a matrix is discussed in Chapter 2. The chapter also discusses, LFP logic together with rank operator (LFP+*rnk*) in relation to *finite model theory* (FMT).

The outline of the Bulatov-Dalmau algorithm is given in Chapter 3. In addition, the chapter provides an illustration of the algorithm of fixed CSP, a system of linear equations, defined over the field of two elements, $\mathbb{F}_2$.

In addition to a summary, Chapter 5 includes some open problems in relation to *few subpowers* [13]. The chapter also discusses how the k-edge operation and its template can be extended to the original work shown in Chapter 4.

Next we present some examples of CSPs in relation to some of the domains mentioned above. However, we shall first define the preliminaries to the CSPs in general, the related relational structures and the ternary template Mal'tsev.

1.1 Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) consists of a set of variables, a set of values for each of the variables and a set of constraints on those variables. Constraints are relations among the variables, in which they restrict the values that these variables may simultaneously be assigned. A solution to a CSP specifies values for all the variables in which the constraints are satisfied. As a result, finding a global assignment to all the variables while satisfying all of the constraints is **NP**-complete. Consequently, it motivates researchers in the field in devising an algorithm where CSPs can be determined in polynomial time.

In general, CSPs can be categorized into two main classes namely *Satisfiability* and *Optimization* type of problems [17]. The former deals with finding an assignment of values to variables that satisfies some constraints. However, assignment of values often require optimal assignment. That is, each assignment of a value to each variable has a related cost or an objective value to it. Optimization type of problems deal with finding such an assignment with the least cost or with the highest objective value. However, often algorithms are designed with the intent purpose for both satisfiability and optimization.

In this writing by constraint satisfaction problem means a finite CSP, in which the constraints are fixed over a finite domain. The goal is to find a value for each of the variables in which all the constraints are satisfied. Hence, if there are n variables, where each variable can take k values in the domain, then there are k^n possible assignments. That is , each variable x_i has a domain A_i, which is the set of possible values that the variable can take. The following definition describes the content of a typical instance of a finite constraint satisfaction problem.

Definition 1.1. A *constraint satisfaction problem* (CSP) contains a finite set of variable, written V, a finite set of values (or domain) for each of the variables and a finite set of constraint, denoted by $\mathcal{C}$, on those variables. Hence, let $V = \{x_1, \dots, x_n\}$, A be a domain of values and $\mathcal{C} = \{C_1, \dots, C_m\}$. In addition, let a CSP instance $\mathcal{P} = (V, A, \mathcal{C})$, in which all constraint relations, denoted by S, belongs to the set of relations Γ.

The problem is whether $\mathcal{P}$ has a solution. The solution to a CSP instance involves a mapping of $f : V \to A$, which satisfies each constraint in $\mathcal{C}$.

The CSP representation allows analysis of the problem structure. That is, CSP states can be defined by values of a fixed set of variables, while goal tests (or allowable values) can be defined by constraints on variable values. The following example is a typical CSP representation of a system of linear equations (algebra).

Example 1.1. A CSP representation on a system of linear equations, over any algebraic field:

$$\begin{array}{lcccccccl}
C_1 : & a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = b_1 \\
C_2 : & a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = b_2 \\
\vdots & & & \vdots & & & & & \vdots \\
C_m : & a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = b_m
\end{array}$$

The problem could then be defined as assigning values to all the variables x_i, where $[n]$ is the set $\{1, \ldots, n\}$ and $i \in [n]$, while satisfying all the constraints C_l, where $l \in [m]$. For instance, the familiar tool for solving system of linear equations is through *Gaussian* elimination. This means there is an algorithm that decides if such a system has a solution in polynomial time [6]. In particular, the algorithm accepts a linear system of size s as input and its running time is bounded by $f(s)$, where f is polynomial. We say that the particular system is an instance of CSP of LINEAR SYSTEM. Although systems of linear equations are in **P**, in contrast there is no known polynomial time algorithm for nonlinear, that is NON-LINEAR SYSTEM. The above computational complexities indicates that certain CSPs are not solvable in polynomial time.

The conjecture presented by Feder and Vardi illustrates that a constraint satisfaction problem over finite domains are either in **P** or is **NP**-complete [9]. We adopt a computational complexity notation from [6] in describing an asymptotic upper bound running time of functions. That is, problems that can be solved in $O(n^k)$, where k is a constant and n is the size of the input, are considered to be in class **P**. However, **NP**-complete problems are very unlikely to be solvable in polynomial time. The work of [12] shows a methodology, by way of polynomial time reductions, in proving a given problem is not solvable in polynomial time. This notion of polynomial time reduction is mostly known as Cook-Levin Theorem [1].

Informally, a problem P_1 is in class **NP**-complete if it is as hard as every other problem P_2 in the class **NP**. A solution to either of these problems in turn can be verified in polynomial time. The existence of intractable cases is a compelling evidence that $\mathbf{P} \neq \mathbf{NP}$. That is, if any **NP**-complete problem is solvable in polynomial time then every problem in **NP** can be solved in polynomial time, which means $\mathbf{P} = \mathbf{NP}$. However, to date no polynomial time algorithm exist for problems of class **NP**-complete. Further, since **NP**-complete is the maximal member of class **NP**, assuming $\mathbf{P} \neq \mathbf{NP}$, then there must be problems in $\mathbf{NP} \setminus \mathbf{P}$ that are not in **NP**-complete. For details of the subject and related proof we refer the reader to [12, 17].

Identifying a problem posed to belong to a certain complexity class is a powerful tool used in searching and or devising an efficient algorithm. In general, algorithms designed for specific problems perform better. We illustrate a CSP instance by posing a simple system of linear equations.

Example 1.2. Consider the following system of linear equations defined over $\mathbb{F}_2$, a field with two elements. Let a finite set of variables, $V = \{x_1, x_2, x_3, x_4\}$ with a finite domain, $A = \{0, 1\}$ and a finite constraint relations, S_i for each line of equation. Let the relation containing all the variables be denoted by R.

$$x_1 + x_3 = 0$$
$$x_2 + x_3 + x_4 = 1$$

Problem: find a relation R that satisfies the entire system simultaneously.

The solution to each constraint, representing each line of equation, is as follows:
$S_1 = \{(x_1, x_3) : (0,0),\ (1,1)\}$; and
$S_2 = \{(x_2, x_3, x_4) : (0,0,1),\ (0,1,0),\ (1,0,0),\ (1,1,1)\}$.

The system can simultaneously be satisfied with the following relations:
$R = \{(x_1, x_2, x_3, x_4) : (0,0,0,1),\ (0,1,0,0),\ (1,0,1,0),\ (1,1,1,1)\}$.

The solution obtained above can easily be verified by substituting the corresponding values into the given system of equations. For instance take $R(x_1, x_2, x_3, x_4) = (0,0,0,1)$ as a solution to the system. That is, by substituting the values into the first and second line of equations, respectively, we get the $L.H.S. \equiv x_1 + x_3 = 0 + 0 = 0 \equiv R.H.S.$; and similarly the $L.H.S. \equiv x_2 + x_3 + x_4 = 0 + 0 + 1 = 1 \equiv R.H.S.$ (L.H.S. and R.H.S. denotes the left and right hand side of an equation, respectively.)

Next we give another CSP example from the propositional logic. That is, let a *boolean* formula φ, consist of the variables $x_1, \ldots, x_n$ and the logical operators $AND(\wedge)$, $OR(\vee)$ and $NOT(\neg)$. For instance, $(x_1 \wedge x_2) \vee (x_2 \wedge x_3)$ is a boolean formula. Hence, let $\varphi(v)$ represent the value of the formula in relation to the variables assigned from the domain A, where $v \in A$. In general, a formula φ is satisfiable if there exists some assignment v such that $\varphi(v)$ is `TRUE`, otherwise the formula is not satisfiable.

Further, a formula φ is said to be in conjunctive normal form (CNF) if it has a conjunction of clauses, where a clause is a disjunction of literals [1]. That is, in logical term, a formula in CNF contains an $AND(\wedge)$ of $OR's(\vee)$ of variables or their negation. In general, a formula in CNF has the conjunction of disjunction of the form: $\wedge_i(\vee_j u_{i_j})$, where u_{i_j} are literals and the terms '$(\vee_j u_{i_j})$' are clauses of the formula with distinct literals. That is, each u_{i_j} is either a variable x_k or its negation $\neg x_k$.

In addition, a kCNF is a CNF formula in which all clauses contain at most k literals [1]. As a result, if we denote all satisfiable CNF formulae as `SAT` problems then, for instance, `3SAT` problem is in reference to all satisfiable 3CNF formulae with 3 distinct literals. The following is an example of a 3CNF formula.

Example 1.3. Let the boolean formula φ be an instance of 3CNF problem with one clause. That is, let φ be defined over the variables $V = \{x_1, x_2, x_3\}$ and the domain $A = \{0, 1\}$, where 0 is `FALSE` and 1 is `TRUE`. Let a finite constraint relations, S_i represent each disjunction of literals (a clause). Hence, let the boolean formula is defined as follows:

$$\varphi = (x_1 \vee \neg x_2 \vee x_3).$$

Problem: is the formula φ satisfiable?

If we consider each clause in a given formula as a relational constraint, S_i then φ is satisfiable if all the clauses are evaluated to `TRUE`. That is, if $S_1 \wedge S_2 \wedge \cdots \wedge S_i =$ `TRUE`.

The solution to the constraint (clause) in the formula φ is as follows:
$S_1 = \{(x_1, x_2, x_3) : (0,0,0),\ (0,0,1),\ (0,1,1),\ (1,0,0),\ (1,0,1),\ (1,1,0),\ (1,1,1)\}$ and hence, φ is satisfiable.

Note that, in the above simple example the formula φ is expressed over a single clause, which in turn contains only three variables . It can easily be verified, say using a truth table, that is requires at most 2^3 steps in verifying if the given formula is satisfiable. However, if a formula in 3CNF is expressed with more than one clause, and possibly more distinct variables between each constraints, verifying the satisfiability of all possible outcomes require much greater than polynomial time.

In general, if a formula φ contains n number of variables, there are 2^n possible assignments. Checking every assignment requires $\Omega(2^n)$ time. As a result, over the length of φ, determining whether the formula is satisfiable does not run in polynomial time [6]. This demonstrates that 3CNF is indeed **NP**-complete. In fact, satisfiability of boolean formulas is **NP**-complete. The polynomial reduction due to Cook and Levin, introduced above, is a useful tool in proving other problems of being **NP**-complete. For instance, satisfiable problems or `SAT` can be reduced to `3SAT`. In turn it can be used to prove to more restricted language of boolean formulas, without restricting the language itself, by considering fewer cases as in satisfiability of 3CNF. For details and proof of reducibility see [1, 6, 12].

Similarly, from the domain of graph theory, *graph colouring* is another example of a widely studied class of satisfiability problem. One such application is the colouring of a geographical map with minimum possible number of colours, in which neighbouring regions would have distinct colours and hence, the constraint. The following is an example of a typical graph colouring problem over a given graph.

Example 1.4. k-colouring problem [6]. Let G be an undirected graph and without any isolated vertices. Let V and E be a nonempty set of vertices and edges, respectively. Hence, for a graph $G = (V, E)$, let $f : V \to C$ is a k-colouring function such that $f(u) \neq f(v)$ for every edge $(u, v) \in E$, where $C = \{1, 2, \dots, k\}$ containing a set of distinct number of colours. The task is to find the minimum number of distinct k colours that can satisfy the constraints imposed on a graph.

1.2 CSP and Relational Structures

Constraint satisfaction problem (CSP) is a relational homomorphism problem [11]. Recall that homomorphism preserves adjacency by mapping vertices between graphs. For instance, consider two graphs G and H. If G is homomorphic to H then there is a mapping of vertices in which $f : V(G) \to V(H)$ such that $f(u)f(v) \in E(H)$, where $uv \in E(G)$. In addition, if a homomorphism of G to H exists then graph G is H-colourable. If graph H is k-colourable then we say G is H-colourable and hence, `GRAPH H-COLOURABILITY`. In fact, homomorphism generalizes graph colourings. As shown in Example 1.4 above, a mapping of vertices of a graph G to k-colouring preserves that $f(u) \neq f(v)$, where $uv \in E(G)$. Therefore, CSP can be treated as a relational homomorphism problem. For more on the complexity of the graph colouring problem, the reader is directed to [11].

The following definition describes a homomorphism between a pair of relational structures $\mathbf{A}$ and $\mathbf{B}$.

Definition 1.2. Let $\mathbf{A} = (A; R)$ and $\mathbf{B} = (B; R')$ be a pair of finite relational structures, where R and R' are of the same type. That is, they both contain the same number of relations of the same arity. A homomorphism,

$$h : \mathbf{A} \to \mathbf{B}$$

is a function which for every k-ary relation symbols R_i in the common type of $\mathbf{A}$ and $\mathbf{B}$ satisfies,

$$(a_1, \ldots, a_k) \in R_i^{\mathbf{A}} \text{ if and only if } (h(a_1), \ldots, h(a_k)) \in R_i^{\mathbf{B}}.$$

We write $\mathbf{A} \to \mathbf{B}$ to indicate that there is a homomorphism $h : \mathbf{A} \to \mathbf{B}$.

In general, given a pair of relational structures, $(\mathbf{A}, \mathbf{B})$ determining the presence of a solution is homomorphism [2, 3]. In which, each relation of $\mathbf{A}$ contain tuples of variables. The corresponding relation $\mathbf{B}$ contain tuples of values in which the variable tuples may take.

Analogous to homomorphism relation, the tuples of variables in $\mathbf{A}$ maps to the tuples of values in $\mathbf{B}$. As a result, we have the following definition of CSP in relation to finite relational structure.

Definition 1.3. Let $\mathbf{A} = (A; R)$ and $\mathbf{B} = (B; R)$ be a pair of finite relational structures, where R is finitely defined over each structures, as in $R_i^{\mathbf{A}}$ and $R_i^{\mathbf{B}}$, respectively. We define the relational CSP over $\mathbf{B}$, CSP($\mathbf{B}$), by:

$$\text{CSP}(\mathbf{B}) = \{\mathbf{A} \mid \mathbf{A} \to \mathbf{B}\}$$

The homomorphic definition of the CSP, as shown above, gives rise to the relational CSP defined over a particular structure. That is, if we consider a homomorphic from G to H as H-colouring, then according to the definition the problem is equivalent to CSP(H) [2]. Similarly, if we consider a `3SAT` problem with a relational structure $\mathbf{A}_{\mathtt{3SAT}} = (A; R_{t_1}, \ldots, R_{t_8})$ we have CSP($\mathbf{A}_{\mathtt{3SAT}}$), where $A = \{0, 1\}$ and $t_1, \ldots, t_8$ are the eight 3-tuples on A.

The Dichotomy Conjecture presented by [9], discussed above, is also applicable to CSPs of relational structures. That is, for any finite structure $\mathbf{B}$, the problem CSP($\mathbf{B}$) is either **NP**-complete or solvable in polynomial time. In addition, the method of polynomial time reduction, discussed above, can also be applied to relational structures.

1.3 Mal'tsev Operation

As discussed above, given two general relational structures, a constraint satisfaction problem is a homomorphism problem among the structures. For instance, the truth assignment in satisfying the 3CNF problem discussed above is homomorphism. That is, each clause requires several ternary relations of boolean values that satisfy the formula, φ.

Let us consider a CSP defined over two general relational structures S and T, in which the homomorphism relation is defined as $f : S \to T$, where T is fixed with a template P. It requires to check, if an input S with template P admits a homomorphism to T [11].

Over the years, researchers in the field have devised several templates in relation to operations like *near-unanimity*, *majority-minority*, *edge*. The edge (or as its known the k-edge) operation is discussed in Chapter 5. In this writing, we consider the ternary operation, *Mal'tsev*. Hence, we give the following definition over a finite set $\mathbf{A}$.

Definition 1.4. Consider a finite set $\mathbf{A}$ and an operation, φ. A 3-ary operation, $\varphi : \mathbf{A}^3 \to \mathbf{A}$, is said to be *Mal'tsev* if it satisfies the following template, such that:

$$\varphi(x, y, y) = \varphi(y, y, x) = x, \text{ for all } x, y \in \mathbf{A}.$$

The use of the Mal'tsev operation over a finite set invariant under φ, satisfying all x, y in $\mathbf{A}$, leads to a class of instances of CSP solvable in polynomial time [3]. That is, the constraint problem over the invariant, $\text{Inv}(\varphi)$ or $\text{CSP}(\text{Inv}(\varphi))$ is tractable. For a detailed proof see [3]. As a result, we assume that Mal'tsev operation, as defined above, preserves the relation.

A complete example of a system of linear equations, over a finite field with two elements $\mathbb{F}_2$, is given in Chapter 3. The example also illustrates the use of the Mal'tsev operation discussed here.

Several real world constraint satisfaction problems are **NP**-complete [1]. As a result, much research effort has been put forth in relation to the conjecture made by Feder and Vardi. In closing the chapter on the preliminaries, we give a list of some known CSP instances of class **P** and **NP**-complete and related areas of domain, respectively. For details see [1, 6, 12].

Example 1.5. CSP instances of tractable and intractable cases.
P, (or polynomial time) problems:

LINEAR SYSTEM - A system of linear equations.

2-COLOURABILITY - A complete bipartite graph $K_{m,n}$, where m and n represents the number of vertices in each set of the partition.

`2SAT` - The set of satisfiability formulas as in 2CNF.

NP-complete problems:

NONLINEAR SYSTEM - A system of nonlinear equations.

3-COLOURABILITY - See Example 1.4 above on k-colouring.

`3SAT` - The set of satisfiability formulas as in 3CNF. See Example 1.3 above.

HAMPATH - A Hamiltonian path in a graph, in which a path visits all vertices exactly once.

kCLIQUE - The problem of determining whether a finite graph contains a clique with at least k vertices.

For more examples, see [1, 6, 12].

Chapter 2

Least Fixed-Point Logic with Rank Operators

In this chapter, we introduce the *least fixed-point* (LFP) logic, with rank (rnk) operators applied on matrices. Further, we discuss the expressiveness of LFP logic with rank operators, LFP+rnk, in comparison to other similar operators like least fixed-point logic with counting, $LFP + C$. The main source for this chapter is, "*Logics with Rank Operators*," by Dawar, Grohe, Holm and Laubner [8].

2.1 LFP+rnk and its relation to First-Order Logic

One of the main open problem in descriptive complexity and database theory is the question whether there exist a logic which captures polynomial time [5, 10]. By this we mean, a logic in which precisely those properties which are decidable in polynomial time can be defined. As an initial point, it is reasonable to assume that such a logic must be an extension of first-order (FO) logic, which also admits recursive definitions. We begin by defining a finite set of relations of a given vocabulary.

Definition 2.1. A *vocabulary* τ is a finite set of relation symbols of $R_1, \ldots, R_n$. Every relation symbol has an *arity* and a natural number associated to it. That is, an m-ary relational symbol R may be denoted by $R(x_1, \ldots, x_m)$.

For simplicity, we assume that a vocabulary can contain constraints and function symbols. Hence, let a vocabulary, containing a finite set of relational symbols, is defined by the set $\tau = \{R_1, \ldots, R_n\}$, where R_i is an m-ary relation symbol. Similarly, let the structure of the vocabulary (τ-structure) be $\mathcal{A} = (A;\ R_1^{\mathcal{A}}, \ldots, R_n^{\mathcal{A}})$, where A is a finite domain set and R_i is an m-ary relation on A, the interpretation of R_i.

We assume the basic familiarity with *first-order* (FO) logic, including the basic notation. See [15] for details on the subject. Now, let S be a k-ary predicate symbol not in the vocabulary, τ. Next, let the first-order formula, in which S is positive and $\bar{X} = (x_1, \ldots, x_k)$, be expressed by the formula $\varphi(\bar{X}, S)$. Assuming that $\mathcal{A}$ is a finite τ-structure, we set the initial of the k-ary predicate S to an empty set. The subsequent k-ary predicate is then determined, knowing the fact that the first-order formula φ, in which the previous k-ary S, is satisfied the structure $\mathcal{A}$ (τ-structure). Therefore, we define the k-ary predicate S, so that $S^0 = \emptyset$ and $S^{i+1} = \{\bar{a} : \mathcal{A} \models \varphi(\bar{a}, S^i)\}$. It is clear that, $S^0 \subseteq S^1 \subseteq \ldots \subseteq S^i \subseteq \ldots \subseteq A$. Since A is a finite set, eventually for some $i \geqslant 0$, we have that, $S^i = S^{i+1}$. Hence, we define S^∞ to be this particular S^i and here we call it the *least fixed-point* of $\varphi(\bar{X}, S)$. As a result, the following holds:

$$\mathcal{A} \models S^\infty(\bar{a}) \text{ if and only if } \varphi(\bar{a}, S^\infty).$$

That is, the least fixed-point obtained by the predicate S (S is not in τ) is satisfied in the structure $\mathcal{A}$ (τ-structure), if and only if, it is defined by the first-order formula, φ. It is easy to see the relation between fixed-point and first-order logic (more on this follows). The following example applied on a graph G, demonstrates the concept we have just discussed. It illustrates, the connection of vertices through edge binary relation, E in relation to the first-order formula, φ defined above.

Example 2.1. Let G be a finite graph with vocabulary τ, containing an edge relation, E. That is, $\tau = \{E\}$. The first-order formula then can be defined as follows:

$$\varphi(x, y, S) : xEy \ \vee \ \exists z(xEz \ \wedge \ S(z, y)).$$

The first-order formula, expresses that vertices x and y are connected through edge binary relation E, or there exist some vertex z connected to x by an edge relation and z is connected to y through binary predicate relation S. Therefore, S^{∞} defines the binary connectedness relation on G and hence the first-order formula, φ. It is a well-known fact that connectedness in graphs, because of the Compactness Theorem, cannot be expressed in first-order logic [15].

The least fixed-point (LFP) shares some properties with first-order logic. It can be generalized to a finite collection of formulas with distinct positive relational symbols. Consider two relational symbols, S and R, not occurring in τ. Next, consider two first-order formulas, namely $\varphi_1(\bar{X}, S, R)$ and $\varphi_2(\bar{X}, S, R)$, in which both relational symbols in the formulas occur positively. Initially, the two relational symbols are empty. By applying simultaneous recursion on both symbols, S and R, we obtain the following:

$$S^0 = \emptyset$$

$$R^0 = \emptyset$$

$$S^{i+1} = \{\bar{a} : \mathcal{A} \models \varphi_1(\bar{a}, S^i, R^i)\}$$

$$R^{i+1} = \{\bar{a} : \mathcal{A} \models \varphi_2(\bar{a}, S^i, R^i)\}$$

It can easily be observed that, for a sufficiently large i (as $i \to \infty$), $S^i = S^{i+1}$ and $R^i = R^{i+1}$. Therefore, S^∞ and R^∞ can be defined as *simultaneous least fixed-points* of $\varphi_1(\bar{X}, S, R)$ and $\varphi_2(\bar{X}, S, R)$, respectively.

Definition 2.2. Suppose, the Least Fixed-Point (LFP) logic consists of relations that are defined as follows. That is, suppose $\varphi_i(\bar{X}, S_1, \ldots, S_k)$, for $i = 1, \ldots, k$, are first-order formulas and positive in $S_1, \ldots, S_k$. Now, let $(S_1^\infty, S_2^\infty, \ldots, S_k^\infty)$ be the simultaneous least fixed-point of first-order formulas of $\varphi_i(\bar{X}, S_1, \ldots, S_k)$. Hence, such fixed-points, $S_1^\infty, \ldots, S_k^\infty$ are in LFP.

As a result, the following theorems captures the relation between first-order logic and least fixed-point, and consequently PTIME.

Theorem 2.1. [16] (1.) first-order logic $\subseteq$ LFP

(2.) LFP $\subseteq$ PTIME

(3.) LFP is closed under the $\wedge$, $\vee$, $\exists$, $\forall$, $\neg$, substitution and relativization.

For example, by (1.), we mean that every problem on finite relational structures that can be expressed in first-order logic can be expressed in the logic LFP, and similarly for (2.).

Theorem 2.2. [14, 18] On ordered structures, LFP = PTIME.

In the late 1980s, as an extension of fixed-point logic, Immerman [14] introduced *fixed-point* logic with *counting* operator, FP+C. It seemed to be a suitable candidate for capturing polynomial time. That is, the least fixed-point logic with counting, LFP+C was promising in capturing polynomial time on several classes of structures including planar graphs, structures of bounded tree width. However, the following is true about LFP+C.

Theorem 2.3. [4] LFP+C does not capture polynomial time on all finite structures.

The result of Theorem 2.3 is deep and relies heavily on the construction of the so called *CFI-property* (after Cai, Fürer and Immerman) and used to distinguish LFP+C from polynomial time. As a result, we define an even more expressive expansion of LFP, the logic LFP+rnk. It provides the capability in computing the rank (rnk) of a given matrix over a general finite field of $\mathbb{F}_p$, defined by a formula over n-tuples, where p is a prime number and n is a positive integer.

Recall, the τ-structure discussed above, is defined by $\mathcal{A} = (A;\ R_1, \ldots, R_m)$. We shall equip the structure with an additional integer sort. As a result, we give a general definition of a two sorted structure as follows.

Definition 2.3. A structure **A** is a *two sorted* structure in a finite vocabulary $\{R_1, \ldots, R_m\}$, where R_i is the k_i-ary relation, if A is the union of two disjoint nonempty sets A_1 and A_2 so that $R_i \subseteq B_1 \times, \ldots, \times B_{k_i}$ where $B_1, \ldots, B_{k_i} \in \{A_1, A_2\}$.

Hence, we define $\mathcal{A}^+$, as an extension of $\mathcal{A}$, by the standard model of arithmetic, such that:

$$\mathcal{A}^+ = (\mathcal{A},\ \omega,\ \{R_i\}_{i \leqslant k},\ +,\ \cdot,\ \leqslant,\ 0,\ 1).$$

In addition, symbols like '$+$' and '$\cdot$' are the usual mathematical operators of addition and multiplication of integers, respectively; and the symbol '$\leqslant$' is the usual linear order. We shall distinguish the variables coming from $\mathcal{A}$-sort and ω-sort. Hence, we will use small letters from the Greek alphabets (like $\alpha, \beta, \gamma, \ldots$) for the latter case, ω-sort. In addition, quantifiers over numerical variables must be bounded in order to prevent undecidability with logical values of 0 (`false`) and 1 (`true`).

Note that since 1 (one) is a constant, every $n \in \omega$ is definable. Although, the logical formulas are dependent on their truth values, which takes values of 0 (`false`) or 1 (`true`). However, it is more convenient to treat the values as numerical terms. As a result, the following definition expresses the concept behind the treatment of the values mentioned.

Definition 2.4. Let $\eta(\bar{x},\ \bar{y})$ be a numerical term defined using both numerical and $\mathcal{A}$-variables (i.e. $\bar{x}$ is a tuple containing variables of both sorts, as is $\bar{y}$).

Hence, for each pair of tuples $\bar{a},\ \bar{b}$, which may contain elements of both sorts, we define: $m_{\bar{a},\bar{b}} = \eta^A(\bar{a}, \bar{b})$. We consider, $M = [m_{\bar{a},\bar{b}}]$, as an integer-valued matrix, whose rows are indexed by $|\vec{x}|$-tuples and columns by $|\vec{y}|$-tuples, respectively. So, for a prime number, p, let M_p be the matrix, such that:

$$M_p = M(mod\, p).$$

That is, we can think of M_p as a matrix over the field $\mathbb{F}_p$ with p elements and $rnk_p M_p$ will be its corresponding rank. Hence, we define LFP+rnk_p to the extension of LFP by the operator rnk_p. Therefore, if we extend LFP by all rnk_p, for $p \geqslant 2$, we obtain the extension LFP+rnk (least fixed-point logic with operator that computes the rank of a definable matrix). The following example, for $p = 2$, proves that LFP+rnk_2 is at least as expressive as LFP+C.

Example 2.2. We show that LFP+rnk_2 is at least as expressive as LFP+C. Let $\phi(\bar{x})$ be a formula whose free variables $\bar{x}$ are of $\mathcal{A}$-sort. In order to determine the number of tuples in $\mathcal{A}$ for which $\phi(\bar{a})$ is true, we consider the following formula:

$$\varphi(\bar{x},\ \bar{y}) : \phi(\bar{x})\ \wedge\ \bar{x} = \bar{y}.$$

Clearly, $rnk_2\ \varphi(\bar{x},\ \bar{y})$ will output the number of all $\bar{a}$ for which $\mathcal{A} \models \phi(\bar{a})$.

Hence, we define the following theorem, concluding the expressive power of LFP+rnk over LFP+C. In addition, we make similar list to Theorem 2.1, in relation to the rank of a matrix.

Theorem 2.4. [8] (1.) LFP+C $\subseteq$ LFP+rnk

(2.) LFP+rnk $\subseteq$ PTIME

(3.) LFP+rnk is closed under the $\wedge$, $\vee$, $\exists$, $\forall$, $\neg$, substitution and relativization

For example, by (1.), we mean that every problem on finite relational structures that can be expressed in logic LFP+C can be expressed in the logic LFP+rnk, and similarly for (2.).

Next, we define a predicate $\Theta(\gamma)$, used in determining the smallest (minimum) value of γ, as shown in equation (4.4). For details and descriptions of γ, see Chapter 4.

Definition 2.5. Let $\Theta(\gamma)$ be a predicate with a free variable, for some $\gamma \in \omega$. Hence, we define $\min\limits_{\gamma} \Theta(\gamma)$, to be the numerical term (function) in which outputs the smallest value of $\gamma \in \omega$ for which $\Theta(\gamma)$ holds.

Therefore, as shown in Theorem 2.1, least fixed-point (LFP) logic is an expressive power extension of first-order (FO) logic. In addition, LFP allows inductive definitions simultaneously, as in recursive functions, while still shares some properties with FO. Note that FO logic uses quantified variables over some non-logical objects. Further, in a topological space, S to contain fixed-point property (FPP), for any continuous function, $f : S \to S$, there must exist s in S, where $f(s) = s$. Thus, the FPP is a topological invariant and expressed by any homomorphism.

2.2 Other Operators

The extension of LFP with rank operators, LFP+rnk, has the property that the truth of a formula in any finite structure can be verified in polynomial time. However, instances of similar complexity are not definable in LFP with counting operators, LFP+C. Further, LFP+rnk is equipped with counting capability the dimension of a definable matrix, over a definable vector space. However, the operation of LFP+C just counts the cardinality of a definable set, in which it is already included in LFP+rnk. As a result, LFP+rnk is a generalization of the counting operations, including the operation through matrix determinants. In addition, whenever the fixed-point operations are not present, the rank operator still remains as expressive, as in system of linear equations (*linear algebra*).

Further, a chosen type of rank (rnk) operator defines the rank of a particular matrix over a finite field $\mathbb{F}_p$, where p is a prime number. The variation over n-tuples and the values of p, where n is a positive integer, leads to several possible expressive power of logic that can be defined, as in LFP+rnk_p. For instance, in Chapter 4, for $p = 2$, the least fixed-point logic together with the rank operator, LFP+rnk_2, is used in counting the nonzero entries (rows) of a given matrix. That is, the rank (rnk_2) is defined over the field $\mathbb{F}_2$ over n-tuples. For more details on the properties of the logic LFP+rnk, see [8].

Chapter 3

Outline of Bulatov-Dalmau's Algorithm

In this chapter, we present the outline of the "Bulatov-Dalmau's" algorithm as depicted in [3]. The algorithm in consideration depends on five main procedures namely: `Solve`, `Next`, `Next-beta`, `Nonempty` and `Fix-values`. The algorithm, together with the sub-procedure calls, provides tractable cases where any finite constraint satisfaction problem (CSP) defined over a finite domain, can be solved in polynomial time. Further, using the procedure `Next`, it extends to handle general cases where any CSP can be solved in polynomial time. Procedure `Next` is designed with the intention that the algorithm would handle any cases. Hence, we defer its details while giving priority to the other remaining procedures.

We shall also maintain similar variables, symbols and relative relationships as shown in [3]. As a result, the procedures are reproduced for discussion purposes. We begin by introducing the preliminaries and some basic background in which the algorithm is expressed. We refer the reader to [3] for correctness and proof of the procedures.

3.1 Preliminaries to the Algorithm

The constraints, for positive integers n, are defined over an n-ary tuple, $\mathbf{t} = (t_1, \ldots, t_n)$ of elements $i_1, \ldots, i_j$ in $[n]$ and $i \in \{1, \ldots, n\}$.

3.1.1 Projection of a Tuple

The *projection* of a tuple over the elements is denoted by $(t_{i_1}, \ldots, t_{i_j})$. In addition, for all n-ary relation R defined over a finite domain A and for all elements in $[n]$, the j-ary relation over the projection R is the set defined by $\{pr_{i_1,\ldots,i_j}\mathbf{t} : \mathbf{t} \in R\}$, where $\mathbf{t}$ is a j-ary tuple. In particular, having $\mathbf{t}$ and $\mathbf{t}'$ of two n-ary tuples, the tuples $(\mathbf{t}, \mathbf{t}')$ witness (i, a, b) if $pr_{1,\ldots,i-1}\mathbf{t} = pr_{1,\ldots,i-1}\mathbf{t}'$, $pr_i\mathbf{t} = a$ and $pr_i\mathbf{t}' = b$, where $(i, a, b) \in [n] \times A^2$.

3.1.2 Signature of Relation

The *signature* of the relation R is the set containing all those relations witnessed by tuples. That is, the signature of R, denoted by $Sig_R \subseteq [n] \times A^2$, is a set containing tuples $(\mathbf{t}, \mathbf{t}')$ that witness (i, a, b), such as:

$$Sig_R = \{(i, a, b) \in [n] \times A^2 : \exists \mathbf{t}, \mathbf{t}' \in R \text{ such that } (\mathbf{t}, \mathbf{t}') \text{ witnesses } (i, a, b)\}.$$

Note that, every relation R has a *compact representation*, denoted by R', if $|R'| \leqslant 2|Sig_R|$. In addition, a compact representation R', in which $R' \subseteq R$, is a representation of R if $Sig_R = Sig_{R'}$. That is, the existence of tuples in R', in which such tuples witness (i, a, b) means such tuples are also in R, and hence its signature.

To illustrate the above, consider the projection of a tuple $\mathbf{e}$ on j (j-ary relation) satisfying:

$$pr_j \mathbf{e}^d_{i,a} = \begin{cases} a & \text{if } i = j \\ d & \text{otherwise, where } d \in A \text{ and } (i,a) \in [n] \times A. \end{cases}$$

That is, the tuples $(\mathbf{e}^d_{i,a}, \mathbf{e}^d_{i,b})$ witnesses (i,a,b), for all $(i,a,b) \in [n] \times A^2$. The compact representation over the relation A^n is the set of tuples $\{\mathbf{e}^d_{i,a} : i \in [n], a \in A\}$. Observe that, a process by the tuples $(\mathbf{t}, \mathbf{t}')$ that witness (i,a,b) requires the ternary operation Mal'tsev on all relations on A invariant under φ.

The following lemma shows that the ternary operation Mal'tsev and the representation of the relation R are the focal points on the construction of the algorithm.

Lemma 3.1. (Bulatov and Dalmau [3]) Let a Mal'tsev operation, over a finite set of domain A be defined as $\varphi : A^3 \to A$. Let R' be a representation of R, where the relation R on A is invariant under φ. Hence, let $\langle R \rangle_\varphi$ be the smallest relation R' that contains R in which invariant under φ. Then $\langle R' \rangle = R$.

The proof to Lemma 3.1 uses the fact that the projection of $\langle R' \rangle$ and R on i, applied inductively for all $i \in \{1, \ldots, n\}$, is equivalent. That is, for each tuple in the relation R, having $(i, pr_i\mathbf{t}, pr_i\mathbf{t}')$ is in Sig_R means it is also in $Sig_{R'}$. See [3] for details.

3.2 Algorithm Detail and its Complexity

The algorithm `Solve`, as proposed in Theorem 1 of [3], decides correctly whether a constraint satisfaction problem instance invariant under φ, CSP(Inv(φ)), has a solution in polynomial time. The input to the algorithm is a CSP(Inv(φ)) instance, equipped with a set of finite variables, $V = \{v_1, \ldots, v_n\}$, a finite set of domain A and a finite set of constraints, $\mathcal{C} = \{C_1, \ldots, C_m\}$.

A typical input $\mathcal{P}$, an instance of CSP, is invariant under φ, where $\mathcal{P} = (V, A, \mathcal{C})$. In addition, the n-ary relation on the domain A is denoted $R_l = \{(s(v_1), \ldots, s(v_n)) : s \text{ is a solution of } \mathcal{P}_l\}$, where $l \in \{0, \ldots, m\}$ and $\mathcal{P}_l = (\{v_1, \ldots, v_n\}, A, \{C_1, \ldots, C_n\})$. As a result, initially $\mathcal{P}_0$ does not have a constraint and similarly for $l = 0$, R_0 is just A^n. Note that, a compact representation R'_0 of R_0 is initially encoded almost everywhere, as illustrated above, with an arbitrary value d an element of the domain. A subsequent compact representation R'_{l+1} of R_{l+1} is obtained iteratively, using the procedure `Next`, in which its input is the current R'_l and the constraint relation S_{l+1}.

The entire algorithm depends on the iterative calls to procedure `Next`, from with in procedure `Solve`. We look into each procedure and its related complexity, independently. Finally, we shall conclude that the algorithm is indeed decides correctly if such a solution is found, in polynomial time. We begin our analysis on the outline of procedure `Solve`.

3.2.1 Procedure `Solve`

The input to the procedure `Solve` is described above. Its output is the compact representation of the relation, R' after m iterations. That is, if R'_m is not empty it returns 'yes' (step 4), otherwise it returns 'no' (step 5), in which the CSP instance has no solution.

```
Procedure Solve($\{v_1, \ldots, v_n\}, A, \{C_1, \ldots, C_m\}$)
Step 1      select an arbitrary element $d$ in $A$
Step 2      set $R'_o := \{\mathbf{e}^d_{i,a} : (i,a) \in [n] \times A\}$
Step 3      for each $l \in \{0, \ldots, m-1\}$ do
                ($let C_{l+1}$ be $((v_{i_1}, \ldots, v_{i_{l+1}}), S_{l+1})$)
Step 3.1        set $R'_{l+1} :=$ Next($R'_l, i_1, \ldots, i_{l+1}, S_{l+1}$)
            end for each
Step 4      if $R'_m \neq \emptyset$ return yes
Step 5          otherwise return no
```

Initially, the compact representation R'_0 (step 2) is set as illustrated above. Further, for each iteration in step 3, subsequent compact representation R'_{l+1} is set to a call to procedure `Next`$(R'_l, i_1, \ldots, i_{l+1}, S_{l+1})$. Note that, the return of the calls made to `Next` is the set, such that:

$$R_{l+1} = \{\mathbf{t} \in R_l : pr_{i_1,\ldots,i_{l+1}}\mathbf{t} \in S_{l+1}\} \text{ where } l = 0, \ldots, m.$$

As a result, the total running time of the algorithm is polynomial and is related to the size of the input. That is, the total running cost of the call is $O(n^9 + (n + |S_{l+1}|)^4|S_{l+1}|n^3)$. For proof of correctness and its related time complexity of the algorithm, which is procedure `Solve`, see [3].

A call to `Next`, in procedure `Solve`, makes subsequent calls to `Nonempty` from procedures

`Next-beta` and `Fix-values`. Hence, next we focus on the procedure `Nonempty`.

3.2.2 Procedure `Nonempty`

This procedure takes a compact representation R', of relation R invariant under φ, a sequence of elements $i_1, \ldots, i_j \in [n]$, (n-arity of R) and a j-ary constraint relation S, also invariant under φ. The procedure returns either an n-ary tuple $\mathbf{t} \in R$, where $pr_{i_1,\ldots,i_j}\mathbf{t} \in S$ or such a tuple does not exist.

```
Procedure Nonempty($R', i_1, \ldots, i_j, S$)
Step 1    set $U := R'$
Step 2    while $\exists \mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3} \in U$ such that $pr_{i_1,\ldots,i_j}\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) \notin pr_{i_1,\ldots,i_j}U$ do
Step 2.1      set $U := U \cup \{\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})\}$
          end while
Step 3    if $\exists \mathbf{t}$ in $U$ such that $pr_{i_1,\ldots,i_j}\mathbf{t} \in S$ then return $\mathbf{t}$
Step 4    else return "no"
```

Initially, the compact representation R', input to the procedure, is set to U (step 1). As a result, the number of iterations (step 2) is bound by the size of U. The cardinality of U increases by one during the iterative loop in step 2. Thus, U is updated (increased) every time some tuples exist in U invariant under φ. Hence, at the end of the execution of the procedure, the size of U is different from step 1. Therefore, the number of iteration can be bounded by the size $|U|$.

The process of checking the existence of some tuples $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}$ in U (step 2), in which the projection of such tuples invariant under φ not in U, is the dominating cost during each iteration. That is, trying all possible combinations of n-tuples found in U, requires $|U|^3$ steps, where each step in turn requires of $O(|U|)$ times. In addition, a sequential search process, in which checking if tuples $\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})$ are already in U, requires $O(|U|)$ times. As a result, step 2 has total running time of $O(|U|^4 n)$. Recall that, this procedure returns a tuple $\mathbf{t}$ in U satisfying $pr_{i_1,\ldots,i_j}\mathbf{t} \in S$ applied on n-ary constraint relations, (step 3). Hence, step 3 has a running time of $O(|U||S|n)$. Therefore, the procedure has a total running time of $O(|U|^5 n + |U||S|n)$, in which it can be bound by $O(|U|^5|S|n)$.

In step 2, as discussed above, the size of the projection of U increases during each iteration. However, it is bounded by the size of the projection of the relation R. Also, recall that the cardinality of the compact representation R' is bounded by $2n|A|^2$ (or $|R'| \leqslant 2|Sig_R|$), where A is a finite domain. Since the size of A is fixed, the cardinality of R' is of $O(n)$. As a result, the procedure can be bounded by $O((n + |pr_{i_1,\ldots,i_j}R|)^5|S|n)$. The output of procedure `Nonempty` plays a major role throughout the algorithm. In particular, procedures `Fix-values` and `Next-beta` depends on the outcome of this procedure, and hence the algorithm. For proof of correctness of procedure `Nonempty` and its related time complexity, see [3].

3.2.3 Procedure `Fix-values`

This procedure, unlike the other procedures, takes two sets of objects. That is, a canonical representation of the relation R invariant under φ and a sequence of elements of A, $a_1, \ldots, a_m$, where $m \leqslant n$ and n is the arity of R. It returns a compact representation of the relation R, in which initially R' is set to U_j (for $j := 0$). At the end of the iteration, procedure

`Fix-values` returns the set U_m such that $\{\mathbf{t} \in R : pr_1\mathbf{t} = a_1, \ldots, pr_m\mathbf{t} = a_m\}$.

```
Procedure Fix-values($R', a_1, \ldots, a_m$)
Step 1     set $j := 0$; $U_j := R'$
Step 2     while $j < m$ do
Step 2.1     set $U_{j+1} := \emptyset$
Step 2.2     for each $(i, a, b) \in [n] \times A^2$ do
Step 2.2.1     if $\exists \mathbf{t_2}, \mathbf{t_3} \in U_j$ witnessing $(i, a, b)$ and
                        (we assume that if $a = b$ then $\mathbf{t_2} = \mathbf{t_3}$) and
                    Nonempty($U_j, j+1, i, \{(a_{j+1}, a)\}$) $\neq$ "no" and
                    $i > j+1$ or $a = b = a_i$ then
                      (let $\mathbf{t_1}$ be the tuple returned by
                                   Nonempty($U_j, j+1, i, \{(a_{j+1}, a)\}$))
                      set $U_{j+1} := U_{j+1} \cup \{\mathbf{t_1}, \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})\}$
               end for each
Step 2.4       set $j := j + 1$
             end while
Step 3       return $U_m$
```

As described above, in step 1 U_0 is set to R', input to the procedure. The iteration starts in step 2 for $m \leqslant n$ times. Further, a sub-loop (step 2.2) follows for each $(i, a, b) \in [n] \times A^2$. Hence, step 2.2 requires a total iteration of $n|A|^2$ times. However, the most dominating cost during each iteration is the call to procedure `Nonempty`, in which tuples $(\mathbf{t_2}, \mathbf{t_3})$ in compact representation U_j (step 2.2.1) that witness (i, a, b). Recall that, procedure `Nonempty` returns a tuple, say $\mathbf{t_1}$, where the projection of the tuple is the constraint relation S. The compact representation then updated using the fact that $\mathbf{t_1}$ and $\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})$ are tuples in the relation R and witnesses (i, a, b). As a result, the cost for each iteration of the loop is $O((n + |A|^2)^4 n) = O(n^5)$. Therefore, the running cost of the procedure can be bounded to $O(n^7)$.

The correctness of the procedure follows from the fact that a compact representation, $U_j = R_j = \{\mathbf{t} \in R : pr_1\mathbf{t} = a_1, \ldots, pr_j\mathbf{t} = a_j\}$, and proved inductively, for $j = 0, \ldots, m$. For complete proof see [3].

3.2.4 Procedure `Next-beta`

This procedure is similar to procedure `Next` and has the worst running time. In addition, it has similar structure to procedure `Fix-values`. It takes the compact representation R', a sequence of elements of $i_1, \ldots, i_j \in [n]$ and a constraint relation S. In relation to its input size, the procedure might have exponential running time. The procedure returns a compact representation U, in which initially set to empty.

```
Procedure Next-beta($R', i_1, \ldots, i_j, S$)
Step 1   set $U := \emptyset$
Step 2   for each $(i, a, b) \in [n] \times A^2$ do
Step 2.1    if Nonempty($R', i_1, \ldots, i_j, i, S \times \{a\}$) $\neq$ "no" then
              (let $\mathbf{t}$ be Nonempty($R', i_1, \ldots, i_j, i, S \times \{a\}$))
Step 2.2       if Nonempty(Fix-values($R', pr_1\mathbf{t}, \ldots, pr_{i-1}\mathbf{t}$),
                              $i_1, \ldots, i_j, i, S \times \{b\}$) $\neq$ "no"
                (let $\mathbf{t}'$ be Nonempty(Fix-values($R', pr_1\mathbf{t}, \ldots, pr_{i-1}\mathbf{t}$),
                              $i_1, \ldots, i_j, i, S \times \{b\}$))
                set $U := U \cup \{\mathbf{t}, \mathbf{t}'\}$
         end for each
Step 3   return $U$
```

The iteration in step 2 processes each tuple in which $(i, a, b) \in [n] \times A^2$. The procedure makes use of the tuples returned by the call to procedures `Nonempty` and `Fix-values` (step 2.2). Recall that procedure `Nonempty` returns a tuple $\mathbf{t}$ in the relation R where the projection of $\mathbf{t} \in S$. In particular, step 2.1 returns a tuple for (i, a) where $pr_i\mathbf{t} = a$. Similarly, step 2.2 returns a tuple $\mathbf{t}'$ in which the projection of $\mathbf{t}' \in S$ for (i, b), where $pr_i\mathbf{t}' = b$. Note that, a sub-call to `Fix-values`, by `Nonempty` in step 2.2, returns the compact representation where $pr_{1,\ldots,i-1}\mathbf{t}' = pr_{1,\ldots,i-1}\mathbf{t}$. Hence, if the condition holds then $(\mathbf{t}, \mathbf{t}')$ witnesses (i, a, b). Otherwise, (i, a, b) is not in Sig_{R^*}, where $R^* = \{\mathbf{t} \in R : pr_{i_1,\ldots,i_j}\mathbf{t} \in S\}$ invariant under φ (discussed in procedure `Next`). See [3] for details and the necessary conditions. That is, conditions for the existence of a tuple $\mathbf{t}'$ in which $(i, a, b) \in Sig_{R^*}$.

As discussed above, procedure `Next-beta` is dependent on the outputs of `Nonempty` and `Fix-values`. As a result, the running cost of `Next-beta` is mainly the consequence of these two procedures, also discussed above. The iteration through each tuple in step 2 is $n|A|$ times. The cost of invoking to procedure `Nonempty` (step 2.1) is $O((n+r)^4|S|n)$, where $r = |pr_{i_1,\ldots,i_j}R|$. However, in step 2.2, `Fix-values` is called from `Nonempty`. Hence, it requires the sum of the calls already established above. As a result, the total running time of procedure `Next-beta` is $O((n+r)^4|S|n^2+n^8)$. Note that, in addition to step 2.1, a call to `Fix-values` (step 2.2), which in turn invokes `Nonempty` (see procedure `Fix-values`). Hence, it is easy to see that procedure `Next-beta` has the worst running time. See [3] for details.

3.2.5 Procedure `Next`

Procedure `Next`, as discussed above, is designed for general cases of constraint satisfaction problems. Hence, we limit the details of this procedure to the extent where the algorithm in discussion is solvable in polynomial time. See [3] for details.

As shown in procedure `Solve` above, `Next` is initially encoded from the compact representation relation R. The input to the procedure includes, the canonical representation R' of relation R invariant under φ, a sequence of elements $i_1, \ldots, i_j \in [n]$ (n-arity of R) and a j-ary relation S, also invariant under φ. It returns the compact representation U, after some iterative calls to procedure `Next-beta`.

```
Procedure Next($R', i_1, \ldots, i_j, S$)
Step 1      set $l := 0$, $U_l := R'$
Step 2      while $l < j$ do
Step 2.1        set $U_{l+1}$ := Next-beta($U_l, i_1, \ldots, i_{l+1}, pr_{i_1,\ldots,i_{l+1}} S$)
            end while
Step 3      return $U_j$
```

The procedure makes a series of calls to `Next-beta` providing U_l, $i_l, \ldots, i_{l+1}$ and its corresponding projection of S, as input. Initially U is set to R' (for $l := 0$). Each iteration in step 2 has the running time of $O(n^8 + (n + |S|)^4 |S| n^2)$, in which r is bounded over a fixed A, where $r = |pr_{i_1,\ldots,i_l} S||A|$. Since the possible set of constraint relation S that can appear in instance $\mathcal{P}$ is infinite, it is difficult to bound the value of j (j-ary relation). As a result, in the worst case the value of $r = |pr_{i_1,\ldots,i_j} R|$ can be exponential. The solution would be to bound the value of j where r can be polynomial over a finite constraints invariant under φ, CSP(Inv(φ)), see [3] for details.

In Chapter 4, similar approach has been applied over a fixed set of constraints. Hence, let m be the number of constraints and S^* be the largest constraint relation occurring in $\mathcal{P}$, instance of CSP invariant under φ. As shown in Corollary 1 ([3]), the algorithm decides correctly where $\mathcal{P}$ is solvable in $O(mn^8 + m(n + |S^*|)^4 |S^*| n^2)$ times. In addition, throughout the algorithm, procedure `Nonempty` plays the centre of decision making process of the existence of tuple $(\mathbf{t}, \mathbf{t}')$ that witnesses (i, a, b).

We have concluded the sketch of the algorithm. As a result, the details of the procedures clearly shows that the algorithm `Solve` is indeed tractable. For details see [3]. We conclude the chapter by providing an illustration of the Bulatov-Dalmau's algorithm.

3.3 Example - Illustration of the Bulatov-Dalmau's Algorithm

Next, we provide a simple example of solving a system of linear equations defined over $\mathbb{F}_2$ (field with two parameters, 0 (`false`) and 1 (`true`)). We assume some basic familiarity with boolean arithmetic over the defined field.

Example 3.1. Consider the following system of three linear equations expressed using four variables, namely x, y, z and u over $\mathbb{F}_2$:

$$\begin{aligned} x + z + u &= 1 \\ y + z &= 0 \\ y + u &= 1 \end{aligned}$$

We remark that for Mal'tsev polymorphism on $\mathbb{F}_2$, we can take, for instance:

$$\varphi(x, y, z) = x + y + z.$$

The above system can be translated into a constraint satisfaction problem (CSP) as a pair of C_i(constraints) and S_i(constraint relations), defined over a finite domain, as:

$$C_1 : \big((x, z, u), \{(1,1,1), (1,0,0), (0,1,0), (0,0,1)\}\big)$$
$$S_1 = \{(1,1,1), (1,0,0), (0,1,0), (0,0,1)\}$$

$$C_2 : \big((y, z), \{(1,1), (0,0)\}\big)$$
$$S_2 = \{(1,1), (0,0)\}$$

$$C_3 : \big((y, u), \{(1,0), (0,1)\}\big)$$
$$S_3 = \{(1,0), (0,1)\}$$

Note that, the number of variables, $n = 4$ and the number of constraints, $m = 3$. The compact representation has a solution, if after some iteration $R'_m \neq \emptyset$. This is achieved using subsequent calls to procedure `Nonempty`, as shown in the algorithm.

Initially, we set almost everywhere to $d = 0$, as shown in procedure `Solve`, to get initial compact representation, R' as:

$$R'_0 = \{(1,0,0,0), (0,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1)\}.$$

Applying iterative process on subsequent relations, we get the next compact representations, as:

$$R'_1 = \{(1,0,0,0), (0,0,1,0), (0,0,0,1), (1,0,0,0)\}$$
$$R'_2 = \{(1,0,0,0), (0,0,0,1), (0,1,1,0)\}$$
$$R'_3 = \{(0,0,0,1), (0,1,1,0)\}$$

R_3' generates the set of all solutions via the Mal'tsev polymorphism. However, it can be easily checked that (0,0,0,1) and (0,1,1,0) are the only solutions and, in fact, the only elements in the subuniverse of $(\mathbb{F}_2)^4$, generated by these two 4-tuples.

Chapter 4

Bulatov-Dalmau Algorithm via LFP+rnk

In this chapter, we translate the Bulatov-Dalmau's algorithm in the logic least fixed-point (LFP) with matrix rank (rnk) operator, LFP+rnk. As a result, we show such CSPs are expressible in logic LFP+rnk providing a logical proof such CSPs are tractable. The procedures mentioned below are referring to the original algorithm as shown in [3]. However, we begin by introducing the necessary variables and the corresponding structure of the language in relation to the fixed constraint problem.

4.1 Preliminaries

Recall that constraint satisfaction problem (CSP) consists a finite set of variables V, a finite set of values A, for each of the variables and a set of constraints C between several collections of variables.

Hence, let $V = \{x_1, \ldots, x_k\}$ and $C = \{C_1, \ldots, C_m\}$ with constraints C_l, so that $l \in [m]$, where $[m]$ denotes the set $\{1, \ldots, m\}$. The constraint scope, $\{x_1, \ldots, x_k\}$, corresponds to k-ary relation on the domain A. Each constraint then contains a tuple of variables and the corresponding values in relation to A.

We assume that the structure $\mathcal{A}$ admits a 3-ary Mal'tsev polymorphism $\varphi(x, y, z)$. In addition, we assume that φ is compatible for all relations in A; namely $S_1, \ldots, S_m$.

Now consider a constraint satisfaction problem $\mathcal{C}$ as a finite "initial segment" of ω over a two-sorted structure defined as $\mathcal{A}^+ = (\omega, A)$, where $\omega = \{+, \cdot, <, 0, 1, rnk_2\}$. An initial segment means that only finitely many elements of ω appear in the constraints. The encoding of the constraint satisfaction problem is given in its general form. That is, as a two sorted structure, $\mathcal{C}$ can be constructed in logspace and therefore, in polynomial time [16]. The rank of a given matrix, rnk_2 is an integer, where $rnk_2 \geqslant 0$ over $\mathbb{F}_2$ (the field with two elements of binary numbers, 0 and 1). Also, A is a finite relational structure which contains the relation $S_1, \ldots, S_k$ and all of its elements are constants. Therefore, the constraint pair of the form $(x_1, \ldots, x_k,\ S_1, \ldots, S_k)$ is a $2k$-ary relation.

We want to construct a sentence (formula without free variables) Φ_A, which can contain constants (parameters) from A, such that $\mathcal{C} \models \Phi_A$ if and only if the instance constraint satisfaction problem coded as $\mathcal{C}$ has a solution. Thus, Φ is a 6-ary formula in the two-sorted language and has the following format:

$$\Phi(\beta, y, z, u; \alpha, x).$$

It contains two variables α and β from the sort of natural numbers and four variables (u, x, y and z) from the sort A, corresponding to the domain A. The formula Φ is a compact matrix representation of the solution $i \in [n]$, where $[n]$ is a positive integer denoting the set $\{1, \ldots, n\}$. In particular, $\Phi(i, a, b, a(\text{or } b); \alpha, x)$, for all $i \leqslant n$, and $a, b \in A$ obtained by replacing β with parameters $1, \ldots, n$ and y, z, u with values from the domain A. That is, either the assigned parameters to y and u are equal or the parameters assigned to z and u are equal. Hence, a witness for (i, a, b) is referring to the i-th coordinate of a matrix in which it contains the value of a or b. Further, a given matrix then contains 1 whenever the α-th coordinate of the tuple is x and 0 otherwise, where α is at most n. The value of n can be obtained by utilizing the counting ability of the logic $LFP + rnk$, as explained in Chapter 2.

In general, $\Phi(i, a, b, a(\text{or } b); \alpha, x)$ is the n-tuple, where a witness for (i, a, b) contains a or b in the i-th coordinate of a given matrix. Thus, for each i-th coordinate of a matrix, having assigned a or b, it requires 2^n overall possible values.

Every time a constraint is processed, Φ is updated accordingly, and produces a compact representation of the relation R. Hence, let R' be the compact representation of the relation, where $R' \subseteq R$. The compact representation for the solution requires the processing of two matrices over the values of the domain A. We define the following constant K, representing the number of 4-tuples of parameters i, a, b, and c, where $c \in \{a, b\}$, such that:

$$K = 2n|A|^2.$$

Hence, the collection of K formulas encodes a compact representation for the solution, R'. Therefore, as defined above, K is the number of 4-tuples that maintains R'.

We shall define a bijection g from all 4-tuples (i,a,b,c), where $c=a$ or $c=b$ to the set $\{1,\ldots,K\}$, in the logic $LFP+rnk$. That is, by using the natural ordering $\leqslant$ of ω and some fixed ordering (for example, lexicographic) of all triples (a,b,c) from $A\times A\times A$, which is independent of n.

Let d be some fixed element in A. Initially, for each matrix we set all those n-tuples which are almost everywhere equal to d, as follows:

$$\begin{array}{c|c}
(a,\ d,\ d,\ \ldots,\ d) & (b,\ d,\ d,\ \ldots,\ d) \\
(d,\ a,\ d,\ \ldots,\ d) & (d,\ b,\ d,\ \ldots,\ d) \\
\vdots & \vdots \\
(d,\ d,\ \ldots,\ a,\ d) & (d,\ d,\ \ldots,\ b,\ d) \\
(d,\ d,\ \ldots,\ d,\ a) & (d,\ d,\ \ldots,\ d,\ b)
\end{array}$$

Similarly, the formula $\Phi(\beta,y,z,u;\alpha,x)$ will consist of, for all values of a and b, if $u=a$, then:

$$\Phi(i,a,b,a;\alpha,x):(\alpha=i\wedge x=a)\vee(\alpha\neq i\wedge x=d) \tag{4.1}$$

and analogously, if $u=b$ where a replaced with b in the formula (4.1).

Now, consider the following function g, an instance of a bijection from ordered pairs $\mathbb{N}\times\mathbb{N}$ to $\mathbb{N}$:

$$g(i,j)=((i+j)(i+j+1)/2)+i.$$

We can iterate the function $g(i,j)$ in order to obtain a bijection between $\mathbb{N}\times\mathbb{N}\times\mathbb{N}$ and

$\mathbb{N}$. Further, we can alter this bijection in order to obtain a bijection f, such that, there is a function f which constitutes a bijection between all ordered triples of elements $\mathbb{N}$ and $\mathbb{N}$. Hence, we get a bijection f, an instance of ordered triples of a bijection between $\mathbb{N}$ and $\mathbb{N}$:

$$f : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \{K+1, K+2, \ldots\}.$$

By iterating this bijection, where K is as described above, we can enumerate the ordered triples in the procedure `Nonempty`.

We are now in a position to implement the procedures as described in Bulatov-Dalmau's algorithm. However, we shall begin by implementing the preliminaries to the procedures.

4.2 Implementation of Supporting Predicates

Recall that, we have a compact representation of the relation R, denoted by R' together with a collection of K tuples as described above. We have also defined some given j integers $i_1, \ldots, i_j$ and some relation S. Implementation of the procedures requires defining two more relations, namely $U(\gamma)$ and $Coord(\gamma; \alpha, x)$.

Hence, $U(\gamma)$ applies to integers while $Coord(\gamma;\ \alpha,\ x)$ applies to integers γ, α and an element $x \in A$. These two relations will define the enumeration process of the tuples that can be constructed from the existing elements of U and the ternary Mal'tsev operation φ. That is, according to the procedure `Nonempty`, $U(\gamma)$ will be true if the tuple indexed by γ winds up being in U. Similarly, the relation $Coord(\gamma; \alpha, x)$ will be true if the α-th coordinate of the tuple indexed by γ is x. Note that in the logic LFP, we can define several relations simultaneously using least fixed-point iteration. See [16] for details of LFP logic in relation to system of recursions.

The relation $U(\gamma)$ is defined using the following facts: $\gamma = 1, \ldots, K$ or there exists $\gamma_1, \gamma_2, \gamma_3$ less than some γ, where such γ is equivalent to value found by $f(\gamma_1, \gamma_2, \gamma_3)$. Further, for all β less than γ, if $U(\beta)$ is true then, for some $\delta \in \{i_1, i_2, \ldots, i_j\}$ and for all $x \in A$, the tuples indexed by β and γ disagree in the δ-th coordinate. That is, a tuple indexed by γ is in U if it is also in R. Thus, either it is indexed (marked) by one of $1, \ldots, K$, or it is generated by some three formulas already in U and it does not agree in coordinates $i_1, \ldots, i_j$ with any of the tuples already in U. The following is then its logical implementation described above:

$$\begin{aligned}
&U(\gamma): \\
&\bigvee_{i=1}^{K} \gamma = i \ \vee\ \exists \gamma_1 \gamma_2 \gamma_3 \Big(\ (\gamma_1 < \gamma) \wedge (\gamma_2 < \gamma) \wedge (\gamma_3 < \gamma) \wedge \gamma = f(\gamma_1, \gamma_2, \gamma_3) \\
&\qquad\qquad \wedge\ \forall \beta < \gamma \Big(\ U(\beta) \rightarrow \exists \delta \bigvee_{l=1}^{j} \delta = i_l \ \wedge\ \forall x \in A \\
&\qquad\qquad\qquad \neg \big(\ Coord(\beta; \delta, x) \wedge Coord(\gamma; \delta, x) \ \big) \ \Big) \ \Big)
\end{aligned} \tag{4.2}$$

The relation $Coord(\gamma; \alpha, x)$ is also defined using the following facts: the coordinate agrees with R if $\gamma = 1, \ldots, K$ and the bijection is equivalent to γ for the first tuples within $1, \ldots, K$. Otherwise, γ is referring to $K+1, K+2, \ldots$ tuples. That is, there exist some $\gamma_1, \gamma_2, \gamma_3$, where the bijection of such γ's is equivalent to γ; and the α-th coordinate of the tuples indexed by such γ's also agree on x. It is obvious that, the value of x must have been obtained by using the ternary operation, φ. Further, the second half of the above statement expresses that, if the tuple indexed by γ is not one of the original $1, \ldots, K$ found in R, then it must be formed using φ from some tuples already in U. The logical format of the statement expressed above follows:

$$
\begin{aligned}
&Coord(\gamma; \alpha, x) : \\
&\quad \Big(\bigvee_{i=1}^{K} \gamma = i \ \wedge \ g(i,a,b,c) = \gamma \rightarrow \Big(Coord(\gamma; \alpha, x) \iff \Phi(i,a,b,c;\alpha,x) \Big) \Big) \\
&\qquad \vee \ \Big(\ \exists\gamma_1 \exists\gamma_2 \exists\gamma_3 \Big(\ U(\gamma_1) \wedge U(\gamma_2) \wedge U(\gamma_3) \wedge \gamma = f(\gamma_1,\gamma_2,\gamma_3) \\
&\qquad\qquad\qquad \wedge Coord(\gamma_1;\alpha,x_1) \wedge Coord(\gamma_2;\alpha,x_2) \\
&\qquad\qquad\qquad\qquad \wedge Coord(\gamma_3;\alpha,x_3) \wedge x = \varphi(x_1,x_2,x_3) \ \Big) \Big)
\end{aligned}
\tag{4.3}
$$

Note that, the formula Φ in equation (4.3) was previously defined above, see equation (4.1), also $c \in \{a, b\}$.

The two relations, namely $U(\gamma)$ and $Coord(\gamma; \alpha, x)$, handles the enumeration process of the tuples as discussed above. Further, the procedures described in the algorithm invokes these two predicates during the process. Next, we implement (translate) the procedures, namely `Solve`, `Nonempty`, `Next-beta` and `Fix-values` in the logic LFP+rnk.

4.3 Implementation of the Procedures via Logic

We begin by implementing the procedure `Solve`. We shall implement this procedure from the fact that Φ_A satisfies over the two sorted structure if and only if the encoded constraint has a solution over the same structure.

`Solve`$_{\{x_1,\ldots,x_n\},A,\{C_1,\ldots,C_m\}}$: it takes an instance of the constraint satisfaction problem, CSP. The CSP contains a list of variables, the value that the variables takes and the list of the constraints. We aim to construct Φ_A, such that:

$$(\omega, A) \models \Phi_A \text{ if and only if CSP encoded as } (\omega, A) \text{ has a solution.}$$

That is, the construction of Φ will satisfy if and only if the constraint over the two sorted structure has a solution. The constraint $\mathcal{C}$, as shown above, is defined over the two sorted structure. In addition, the compact representation of the relation R, denoted by R', contains almost everywhere $d \in A$. That is, initial R'_0 is encoded as $(i, a) \in [n] \times A$, where the rest is d. Hence, let m be the number of constraints, in which it can be obtained by counting all distinct k_j-tuples in the numerical (constant) variables that appear during the encoding of the constraints. As shown above, the conjunction of the formula Φ_A counts the value of m for which the rank of the matrix, rnk_2 is greater than 0. That is, there exists a β up to including K, that satisfies the formula Φ_A, where the α-th coordinate is x. That is, $\exists \beta \leqslant K\ \exists y \exists z \exists u\ rnk_2((R, m)(\beta, y, z, u; \alpha, x)) > 0$. Further, for $k = 0, 1, \ldots, m$ the next $(R, k)(\beta, y, z, u; \alpha, x)$ can be recursively defined as follows:

- for $k = 0$, $(R, 0)(\beta, y, z, u; \alpha, x)$ is the disjunction of matrix formulas which define n-tuples, which are almost everywhere d; and
- for $k = k + 1$, we have $(R, k + 1)$, such as:
  - an input to the procedure: `Next-beta`$_{((R,k),i_1,\ldots,i_{k+1},s_{k+1})}(\beta, y, z, u; \alpha, x)$; and
  - a subsequent constraint, where $C_{k+1} = ((x_{i_1}, \ldots, x_{i_{k+1}}), S_{k+1})$.

Note that, S_{k+1} is first-order definable from the encoding of the constraint C_{k+1} within the structure (ω, A). Hence, $rnk_2(\texttt{Nonemty}_{R',i_1,\ldots,i_j,S}(\alpha, x)) > 0$ if and only if the matrix-construct, Φ_A contains a non-zero rows in which the encoded CSP has a solution. Similarly, at the end of the iteration where $k = m$, if R'_m is not empty.

We also need to check whether there are tuples in U, which are also in the constraint relation S, when projected on to $i_1, \ldots, i_j$. It is clear that, using the ordering (sort) of $\mathbb{N}$, we can find M, the maximum γ, for which the relation $U(\gamma)$ holds. The procedure `Nonempty` checks if a witness to such tuple exists. If such a tuple does not exist, this particular tuple will be filled with `0(false)` at every entry of the matrix.

Next, we implement the procedure `Nonempty`, for which the tuple witness the above, by defining a new relation formula Θ.

$\texttt{Nonempty}_{R',i_1,\ldots,i_j,S}(\alpha, x)$: it takes the compact representation of the relation R, sequence of elements $\{i_1, \ldots, i_j\}$, and a constraint relation S, where the element of the α-th coordinate is x. It will be of the form:

$$\Theta_1 \wedge \Theta_2 \tag{4.4}$$

The definition of each of the relation formulas in equation (4.4) follows:

$$\Theta_1 : \ \ \exists \gamma \leqslant M \ \ \Theta(\gamma) \rightarrow \gamma_0 = \underset{\gamma}{min} \Theta(\gamma) \wedge Coord(\gamma_0; \alpha, x), \text{ and}$$

$$\Theta_2 : \neg \exists \gamma \leqslant M \ \ \Theta(\gamma) \rightarrow (\alpha \neq \alpha \wedge x \neq x).$$

The formula $\Theta(\gamma)$ in the above formulas is the relation defined by:

$$\Theta(\gamma) : \exists i_1, \ldots, i_k \ \exists x_1, \ldots, x_j \ \big(\ U(\gamma) \ \wedge \ Coord(\gamma; i_k, x_k) \ \wedge \ S(x_1, \ldots, x_j)\ \big).$$

The formula Θ_1 defines the existence of a γ fewer than M, where γ_0 is the minimal for which the assumption (witness) holds, with an element x in the α-th coordinate. That is, we select the one with smallest index for tuple that witness this. Similarly, the formula Θ_2 defines that if the relation defined by Θ_1 fails (`false`), then it is false for all α and x. The formula Θ is then verifies the existence of such γ for the element x.

`Next-beta`$_{R', i_1, \ldots, i_j, S}$: it takes compact representation R, sequence of elements $i_1, \ldots, i_j \in [n]$ and the constraint relation S; and produces the updated K formulas for the compact representation.

Now, let the compact representation, Ψ for the relation which consists of all the tuples in R, which have $a_1, \ldots, a_m$ in its first m coordinates, be defined by $\Psi(m, i, a, b, a(\text{or } b); \alpha, x)$. Also note that, for a given matrix P, a_i is the element x for which $P(i, x)$, where $i = 1, \ldots, m$. Hence, we construct the formula $\Psi(k, i, a, b, a(\text{or } b); \alpha, x)$ for $k = 0, 1, \ldots, m$.

Note that, for $k = m$, the formula Ψ is simply Φ. Initially, U_0 is equivalent to Φ and R'. That is $\Psi(0, i, a, b, a(\text{or } b); \alpha, x)$ will be $\Phi(i, a, b, a(\text{or } b); \alpha, x)$, in which $U_0 = \Phi$ ($l := 0,\ U_l := R'$). The next Ψ_{j+1} is expressed from the fact that, if $\Psi(j, i, a, b, a(\text{or } b); \alpha, x)$

is defined (`true`) then $\Psi(j+1, i, a, b, a(\text{or } b); \alpha, x)$, for $0 \leqslant j \leqslant m$, will also be true as follows:

$$\begin{aligned}
&\forall j \leqslant m \ \ \forall i \leqslant n \ \ \forall \alpha \leqslant n \ \ \forall a, b, x, y, z \in A \\
&\qquad\qquad \Psi(j, i, a, b, a; \alpha, x) \wedge \Psi(j, i, a, b, b; \alpha, y) \\
&\qquad\qquad\qquad\qquad \wedge\, rnk_2\big(\mathtt{Nonempty}_{\Psi_j, j+1, i, \{(a_{j+1},\, a)\}}(\alpha, z)\big) > 0 \ \wedge\ \big(i > (j+1) \ \vee\ a = b = a_i\big) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \to \Psi(j+1, i, a, b, a; \alpha, z) \wedge \Psi\big(j+1, i, a, b, b; \alpha, \varphi(z, x, y)\big)
\end{aligned}$$

The output of the procedure is then the compact representation of the K formulas. Note that, the above formula also includes procedure `Fix-values`. In which, it takes the compact representation of the relation R, originated from the K tuples over the formula Φ and an m-tuples ($m \leqslant n$) encoded matrix with elements from the domain A.

Hence, the constructed formula Φ will be satisfied in A if and only if the constraint $\mathcal{C}$ over the two sorted structure have a solution. Initially, the compact representation of the relation R'_0, encoded as $(i, a) \in [n] \times A$, contains almost everywhere $d \in A$. The number of constraints m, can be obtained by counting all distinct k_j-tuples that appear during the encoding of the constraints. That is, the conjunction of the formula Φ_A counts the value of m for which the rank of the matrix is greater than 0. Further, for $k = 0, 1, \ldots, m$ and for β fewer than K, the next constraint relation is defined using $(R, k)(\beta, y, z, u; \alpha, x)$, that satisfies the formula Φ_A, in which the α-th coordinate is x. As a result, R'_m will have a solution if and only if the matrix-construct, Φ_A contains a non-zero (`true`) rows.

Therefore, we have shown constraint satisfaction problems defined over a finite Mal'tsev operation are expressible in the logic LFP+rnk. We have also shown a logical proof that such

CSPs are solvable in polynomial time. Hence, we conclude that Bulatov-Dalmau's algorithm can be formulated in the logic LFP+rnk.

Chapter 5

Conclusion and Open Problems

We conclude by providing the summary on the materials discussed so far and open problems in relation to the subject area, constraint satisfaction problem (CSP). Hence, we begin by highlighting the major points discussed in the previous chapters.

5.1 Summary

In Chapter 1, we started by defining the concept behind the constraint satisfaction problem over a finite relational template. We also illustrated the concept by presenting several examples.

The content of Chapter 2 was mainly concerned with the introduction of a natural extension of first-order (FO) logic. In particular, the chapter is more focussed on least fixed-point (LFP) logic, which can express the concept of recursively definable relations on finite structures. In addition to recursion, we gave a basic overview of a further expansion of the expressive power by introducing a two-sorted logic, LFP+rnk. Further, the expansion allows for basic matrix arithmetic over finite fields, including the computation of the rank function (or rank of a matrix).

The outline of the so-called Bulatov-Dalmau algorithm, for solving constraint satisfaction problems on finite templates with Mal'tsev polymorphism, was given in Chapter 3. The exposition of this chapter, in turn, summarizes the results of Bulatov-Dalmau [3]. In addition, at the end of Chapter 3, we provided an example of the application of Bulatov-Dalmau's algorithm on a system of linear equations defined over $\mathbb{F}_2$.

Finally, by putting all of this together, in Chapter 4 we have demonstrated the original work. As a result, we have given the proof that Bulatov-Dalmau algorithm can be formulated in the logic LFP+rnk. Thus, providing a logical proof that Mal'tsev templates gives rise to tractable constraint satisfaction problems. As a result, we concluded the chapter, in which a formula Φ_A in LFP+rnk such that, $\mathcal{A} \models \Phi_A$ if and only if CSP(A) is in **P**. Also note that, it is not known if such formulas exist whenever A is not Mal'tsev or "bounded width".

5.2 Open Problems

The open problems discussed is in relation to the constraint satisfaction problems presented in this writing. That is, a finite CSP invariant over the Mal'tsev operation. Hence, we look into other templates as a natural extension for further research area. The work of [13] shows the application of a k-edge operation (defined below) on Bulatov-Dalmau's algorithm. As a result, we rely on [13] as a primary source over the relational template discussed above. See [13] for further details.

Definition 5.1. ([13]) Let a k-edge operation φ on a finite template A be the identities, such that:

$$\begin{aligned}
\varphi(x,x,y,\ldots,y) \approx \varphi(x,y,x,y,\ldots,y) &\approx y,\\
\varphi(y,y,y,x,y,\ldots,y) \approx \varphi(y,y,y,y,x,y,\ldots,y) &\approx \cdots\\
\cdots \approx \varphi(y,y,y,\ldots,y,x) &\approx y.
\end{aligned}$$

If a finite relational template A admits a polymorphism, a k-edge operation for some $k > 1$, then we say that A has *few subpowers*.

The Mal'tsev operation (defined in Chapter 1) and near-unanimity are special instances of k-edge operations. In general, a k-edge operations requires a $k+1$-ary operations satisfying the identities shown above. A recent algebraic result establishes that every template with few subpowers is tractable.

Theorem 5.1. ([13]) For any finite relational templates A, if A has few subpowers CSP(A) is in **P**.

The proof of this deep fact relies on the fact that Bulatov-Dalmau algorithm can easily be adopted. Recall that, procedure `Fix-values` takes pair of objects as input (see Chapter 3). The compact representation can still be maintained, while only modifying the `Fix-values` in order to accommodate the changes, template A.

Thus, we strongly believe that the answer to the following problem is affirmative.

Problem 5.1. Is the CSP(A) for any finite template with few subpowers expressible in the logic LFP+rnk?

A more general problem is associated with the main open problem in the area, the so-called *Dichotomy Conjecture*, due to Feder and Vardi.

Conjecture. (Dichotomy Conjecture [9]) Every CSP with a finite relational template is either in **P** or it is **NP**-complete.

To date, all available evidence points to the affirmative answer to this conjecture.

Problem 5.2. Is every CSP which is not **NP**-complete expressible in logic LFP+rnk?

If the answer to Problem 5.2 is "yes", then the Dichotomy Conjecture holds. That is, every CSP expressible in LFP+rnk is necessarily tractable.

NP-complete CSPs with finite relational templates can be characterized algebraically. That is, CSPs whose templates admit a particular algebraically weak polymorphism, can be reformulated algebraically as the following problem.

Problem 5.3. Is the CSP for any finite relational template admitting a weak unanimity polymorphism, in which a polymorphism with the property such that:

$$w(xx \dots xxy) = w(xx \dots xyx) = \dots = w(yx \dots xxx),$$

definable in LFP+rnk?

Bibliography

[1] S. Arora and B. Barak, *Complexity: A Modern Approach*, Cambridge University Press (2009).

[2] A. Bulatov, *Mal'tsev constraints are tractable*, Technical report PRG-RR-02-05, Oxford University Computing Laboratory (2001).

[3] A. Bulatov and V. Dalmau, *A simple algorithm for Mal'tsev constraints*, SIAM J. Comput. **36**(1) (2006), 16–27.

[4] J. Cai, M. Fürer and N. Immerman, *An optimal lower bound on the number of variables for graph identification*, Combinatorica **12**(4) (1992), 389–410.

[5] A. Chandra and D. Harel, *Structure and complexity of relational queries*, J. Compute. Syst. Sci. **25** (1982), 99–128.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms* (2nd Ed.), Massachusetts Institute of Technology (2007).

[7] N. Creignou, P. Kolaitis and H. Vollmer (Eds.), *Complexity of Constraints: An Overview of Current Research Themes*, LNCS 5250, Springer (2008).

[8] A. Dawar, M. Grohe, B. Holm and B. Laubner, *Logics with rank operators,* Proceedings of the 24th IEEE Symposium on Logics in Computer Science, IEEE Computer Science Press (2009).

[9] T. Feder and M. Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory,* SIAM J. Comput. **28**(1) (1998), 57–104.

[10] Y. Gurevich, *Logic and the challenge of computer science*, in E. Börger (Ed.), Current Trends in Computer Science, Computer Science Press (1988), 1–57.

[11] P. Hell and J. Nešetřil, *Graphs and Homomorphisms*, Oxford Lecture Series in Mathematics and its Applications 28, Oxford University Press (2004).

[12] J. Hopcroft, R. Motwani and J. Ullman, *Introduction to Automata Theory, Languages and Computation* (3rd Ed.), Pearson (2006).

[13] P. Idziak, P. Marković, R. McKenzie, M. Valeriote and R. Willard, *Tractability and learnability arising from algebras with few subpowers*, SIAM J. Comput. **39**(7) (2010), 3023–3037.

[14] N. Immerman, *Expressibility as a complexity measure: results and directions*, in Second Structure in Complexity Theory Conference (1987), 194–202.

[15] C. C. Leary, *A Friendly Introduction to Mathematical Logic*, Prentice Hall (2000).

[16] L. Libkin, *Elements of Finite Model Theory*, Texts in Theoretical Computer Science (an EATCS Series), Springer (2010).

[17] D. Poole, A. Mackworth and R. Goebel, *Artificial Intelligence: A Logical Approach*, Oxford University Press (1998).

[18] M. Y. Vardi, *The complexity of relational query languages*, in Proc. of the 14th ACM Symp. on the Theory of Computing (1982), 137–146.