A SIMPLE ALGORITHM FOR MAL'TSEV CONSTRAINTS

ANDREI BULATOV[1] AND VÍCTOR DALMAU[2]

[1]School of Computing Science, Simon Fraser University
8888 University Drive, Burnaby, V5A 1S6, BC, Canada
abulatov@cs.sfu.ca

[2]Departament de Tecnologia, Universitat Pompeu Fabra
Estació de França, Passeig de la circumval.lació, Barcelona 08003, Spain
victor.dalmau@upf.edu

Abstract. A Mal'tsev operation is a ternary operation φ that satisfies the identities $\varphi(x, y, y) = \varphi(y, y, x) = x$. Constraint satisfaction problems involving constraints invariant under a Mal'tsev operation constitute an important class of CSP, which includes affine satisfiablity problem [21], subgroup and near subgroup constraints [13, 12] and many others. It is shown in [5] that any tractable case of the Counting Constraint Satisfaction problem involves only Mal'tsev constraints.

The first algorithm solving arbitrary constraint satisfaction problem with Mal'tsev constraints has been given in [2]. However, this algorithm is very sophisticated and heavily uses advanced algebraic machinery. In this paper, we give a different and much simpler algorithm for this type of contraints.

1. Introduction

Constraint satisfaction problems arise in a wide variety of areas, such as combinatorics, logic, algebra, and artificial intelligence. An instance of the constraint satisfaction problem (CSP) consists of a set of variables, a set of values (which can be taken by the variables), called a domain, and a set of constraints, where a constraint is a pair given by a list of variables, called the constraint scope, and a relation indicating the valid combinations of values for the variables in the scope, called the constraint relation; the goal in a CSP is to decide whether or not there is an assignment of values to the variables satisfying all of the constraints. It is sometimes customary to cast the CSP as a relational homomorphism problem [13], namely, the problem of deciding, given a pair $(\mathbf{A}, \mathbf{B})$ of relational structures, whether or not there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$. In this formalization, each relation of $\mathbf{A}$ contains tuples of variables that are constrained together, and the corresponding relation of $\mathbf{B}$ contains the allowed tuples of values that the variable tuples may take.

The CSP is NP-complete in general, motivating the search for polynomial-time tractable cases of the CSP. A particularly useful way to restrict the CSP in order to obtain tractable cases is to restrict the types of constraints that may be expressed, by requiring the relations appearing in a constraint to belong to a given fixed set Γ. Such a restricted version of the CSP is denoted by CSP(Γ). This form of restriction can capture and place into a unified framework many particular cases of the

CSP that have been independently investigated – for instance, the Horn Satisfiability, 2-Satisfiability, and Graph H-Colorability problems. Schaefer was the first to consider the class of problems CSP(Γ); he proved a now famous dichotomy theorem, showing that for every set Γ of relations over a two-element domain, CSP(Γ) is either solvable in polynomial time, or NP-complete [21].

In recent years, much effort has been invested towards the program of isolating all sets Γ of relations (or *constraint languages*) over a finite domain, that give rise to a class of instances of CSP, CSP(Γ), solvable in polynomial time. Impressive progress has been made along these lines leading to the identification of several broad conditions on Γ that guarantee tractability. It was initiated by the papers [15, 16, 8], where it has been shown that the complexity of CSP(Γ) depends only on the *algebraic invariance properties* of relations from Γ.

Some important recent achivements in the field include a complete classification of CSP problems over a 3-element domain [1] and the conservative CSP [3]. Remarkably, one of main ingredients in both results is the recent result due to the first author [2] stating that every set Γ of relations on a finite set invariant with respect to a Mal'tsev operation, that is, a ternary operation φ satisfying $\varphi(x, y, y) = \varphi(y, y, x) = x$ for all x, y, gives rise to a tractable problem class. This result also encompasses and generalizes several previously known tractable cases of the CSP, such as affine problems [15, 21], constraint satisfaction problems on finite groups with near subgroups and its cosets [13, 12], and paraprimal CSP [10].

Another reason why Mal'tsev constraints are so important is that, to date, almost all CSPs known to be solvable in polynomial time can be solved using *local propagation algorithms*. In this type of algorithms we identify forbidden combinations of values for sets of variables of fixed size and then propagate the original problem by imposing new constraints that use this information. The only known exception is CSPs involving constraints invariant under a Mal'tsev operation. Thus, Mal'tsev constraints constitute an conceptually important class of constraints that require completely different approach in solution techniques. Feder and Vardi attempted to capture this distinction in [13], where they used Datalog programs to simulate local propagation algorithms and suggested the term *problems with ability to count* for those problems which cannot be solved using Datalog. We also note another appearence of Mal'tsev constraints [5]. In that paper, we show that invariance with respect to a Mal'tsev operation is a necessary condition for the tractability of the Counting CSP.

It is fair to say that the original proof of the tractability of Mal'tsev constraints [2] is very complicated. Furthermore, it makes intensive use of advanced algebraic techniques and so is hardly comprehensible for a non-algebraist. In this paper we give a different proof of the tractability of Mal'tsev contraints. The proof presented in this paper is notably simpler than the original proof and does not require the use of any previous algebraic result; indeed the proof is completely self-contained.

2. Preliminaires

2.1. **Constraint Satisfaction Problem.** Let A be a finite set and let n be a positive integer. A n-ary relation on A is any subset of A^n. In what follows, for every positive integer n, $[n]$ will denote the set $\{1, \ldots, n\}$.

A constraint satisfaction problem is a natural way to express simultaneous requirements for values of variables. More precisely,

Definition 1. An instance $\mathcal{P} = (V; A; \{C_1, \ldots, C_m\})$ of a *constraint satisfaction problem* consists of:

- a finite set of variables, $V = \{v_1, \ldots, v_n\}$;
- a finite domain of values, A;
- a finite set of constraints $\{C_1, \ldots, C_m\}$; each constraint $C_l,\ l \in [m]$ is a pair $((v_{i_1}, \ldots, v_{i_{k_l}}), S_l)$ where:
  - $(v_{i_1}, \ldots, v_{i_{k_l}})$ is a tuple of variables of length k_l, called the *constraint scope* and
  - S_l is an k_l-ary relation on A, called the *contraint relation.*

A *solution* to a constraint satisfaction problem instance is a mapping $s : V \to A$ such that for each constraint C_l, $l \in [m]$, we have that $(s(v_{i_1}), \ldots, s(v_{k_l})) \in S_l$.

The question in the CSP instance $\mathcal{P}$ is whether or not there exists a solution to $\mathcal{P}$. $\square$

The constraint satisfaction problem is NP-complete in general, even when the constraints are restricted to binary constraints [19] or when the domain of values has size 2 [9].

Example 1. An instance of the standard propositional 3-SATISFIABILITY problem [14, 20] is specified by giving a formula of propositional logic consisting of a conjunction of clauses, each of which contains exactly 3 literals, and asking whether there are values for the variables which make the formula true.

Suppose that $\Phi = F_1 \wedge \cdots \wedge F_n$ is such a formula, where the F_i are clauses. The satisfiability question for Φ can be expressed as the constraint satisfaction problem instance $(V, \{0, 1\}, \mathcal{C})$, where V is the set of all variables appearing in the clauses F_i, and $\mathcal{C}$ is the set of constraints $\{(s_1, R_1), \ldots, (s_n, R_n)\}$, where each constraint (s_k, R_k) is constructed as follows:

- $s_k = (x_1^k, x_2^k, x_3^k)$ where x_1^k, x_2^k, x_3^k are the variables appearing in clause F_k;
- $R_k = \{0, 1\}^3 \setminus \{(a_1, a_2, a_3)\}$ where $a_i = 1$ if x_i^k is negated in F_k and $a_i = 0$ otherwise (i.e., R_k contains exactly those 3-tuples that make F_k true).

The solutions of this instance are exactly the assignments which make the formula Φ true. $\square$

Example 2. An instance of GRAPH k-COLORABILITY problem consists of a graph G. The question is whether the vertices of G can be labelled with k colours so that adjacent vertices are assigned different colours.

This problem can be expressed as a CSP instance as follows. Let $G = (V, E)$ be a graph. Then we treat V as the set of variables (thus interpreting vertices of G as variables). For the domain we use a k-element set A of colours. Finally, for each edge $(u, v) \in E$ we introduce a constraint $((u, v), \neq_A)$, where $\neq_A$ is the disequality relation on A, defined by

$$\neq_A \ = \{(a, b) \in A^2 \mid a \neq b\}.$$

In applications of the CSP, we normally need just some restricted versions of the problem. One of the most natural and useful ways to restrict the CSP is to impose restrictions on the allowed constraint relations.

Definition 2. For any set of relations Γ, CSP(Γ) is defined to be the class of decision problems with:

- Instance: A constraint satisfaction problem instance $\mathcal{P}$, in which all constraint relations are elements of Γ.
- Question: Does $\mathcal{P}$ have a solution?

Example 1 (continued). If we define $\Gamma_{\text{3-SAT}}$ to be the constraint language on $\{0,1\}$ consisting of all relations expressible by 3-clauses, then any instance of 3-SATISFIABILITY can be expressed as an instance of CSP($\Gamma_{\text{3-SAT}}$) and vice versa. In other words, 3-SATISFIABILITY is equivalent to CSP($\Gamma_{\text{3-SAT}}$). □

Example 2 (continued). Similarly, the GRAPH k-COLORABILITY can be viewed as CSP($\{\neq_A\}$). □

In the two examples we have given, even the restricted problems are NP-complete. However, in many cases restricting the allowed form of constraint relations we arrive to a problem solvable in polynomial time (we refer to such problems as to *tractable* problems).

Example 3. An instance of GRAPH UNREACHABILITY consists of a graph $G = (V, E)$ and a pair of vertices, $v, w \in V$. The question is whether there is no path in G from v to w.

This can expressed as the constraint satisfaction problem instance $(V, \{0,1\}, \mathcal{C})$ where

$$\mathcal{C} = \{(e, \{=_{\{0,1\}}\}) \mid e \in E\} \cup \{((v), \{(0)\}), ((w), \{(1)\})\}$$

where $=_{\{0,1\}}$ denotes the equality relation on the set $\{0,1\}$.

If we define Γ_{UNREACH} to be the constraint language on $\{0,1\}$ containing just the relations $=_{\{0,1\}}$, $\{(0)\}$ and $\{(1)\}$, then any instance of GRAPH UNREACHABILITY can be expressed as an instance of CSP(Γ_{UNREACH}), in this way. □

The research project, which this paper is a part of, aims to distinguish those constraint languages Γ which give rise to a tractable problem CSP(Γ), from those which do not.

In the CSP literature, it is usual to assume constraint languages to be finite. This motivated by certain difficulties of representing infinite collections of relations, by the fact that applications of the CSP in discrete mathematics normally require only some fixed variety of relations, and also by some traditions well established in the area. The algebraic approach we introduce in the next section makes it very natural to deal with infinite constraint languages. Thus, in this paper, the constraint languages we consider are allowed to be infinite. This implies, in particular, that the arity of relations is not necessarily bounded.

Example 4. Let A be a finite field, and let Γ_{LIN} be the constraint language consisting of all relations over A which consist of all solutions to some linear equation over A. Any relation from Γ_{LIN}, and therefore any instance of CSP(Γ_{LIN}), can be represented by a system of linear equations over A (for more details, see [7]).

Clearly, Γ_{LIN} is infinite and yet every instance of CSP(Γ_{LIN}) can be easily defined and it can be solved in polynomial time (e.g., by Gaussian elimination). □

2.2. **Polymorphisms and Invariants.** We briefly introduce the basics of the algebraic approach to the CSP. For more details the reader is referred to [8].

Definition 3. Let $\varphi : A^m \to A$ be an m-ary operation on A and let R be an n-ary relation over A. We say that R is *invariant* under φ and φ is said to be a *polymorphism* of R if for all (not necessarily different) tuples $\mathbf{t_1} = (t_1^1, \ldots, t_n^1), \ldots, \mathbf{t_m} =$

$(t_1^m, \dots, t_n^m)$ in R, the tuple $\varphi(\mathbf{t_1}, \dots, \mathbf{t_m})$ defined as

$$(\varphi(t_1^1, \dots, t_1^m), \dots, \varphi(t_n^1, \dots, t_n^m))$$

belongs to R.

Let C be a set of operations on A and Γ a constraint language. Then $\mathrm{Inv}(C)$ denotes the set of all relations invariant under each operation from C; and $\mathrm{Pol}(\Gamma)$ denotes the set of all operations which are polymorphisms of every relation from Γ.

Example 5. Let R be the solution space of a system of linear equations over a field F. Then the operation $\varphi(x, y, z) = x - y + z$ is a polymorphism of R. Indeed, let $A \cdot \mathbf{x} = \mathbf{b}$ be the system defining R, and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in R$. Then

$$A \cdot \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z}) = A \cdot (\mathbf{x} - \mathbf{y} + \mathbf{z}) = A \cdot \mathbf{x} - A \cdot \mathbf{y} + A \cdot \mathbf{z} = \mathbf{b} - \mathbf{b} + \mathbf{b} = \mathbf{b}.$$

In fact, the converse can also be shown: if R is invariant under φ then it is the solution space of a certain system of linear equations. □

The cornerstone theorem proved by Jeavons et al. [15, 16] provides a link between the complexity of $\mathrm{CSP}(\Gamma)$ and the properties of the polymorphisms of Γ. It amounts to say that, for every *finite* constraint language Γ, the complexity of $\mathrm{CSP}(\Gamma)$ depends solely on the set $\mathrm{Pol}(\Gamma)$. In other words, if $\mathrm{Pol}(\Gamma_1) = \mathrm{Pol}(\Gamma_2)$ then $\mathrm{CSP}(\Gamma_1)$ and $\mathrm{CSP}(\Gamma_2)$ are polynomial time reducible to each other (provided Γ_1 and Γ_2 are finite). This also motivates the prior study of constraint languages of the form $\mathrm{Inv}(C)$ for some set C of operations. In fact, most of the existing tractability results on constraint languages may be formulated as the tractability of problems $\mathrm{CSP}(\mathrm{Inv}(C))$, where C consists of a single operation! (See, e.g. [16, 17, 11, 6, 18].)

One such a type of operations that guarantees the tractability of the corresponding CSP is Mal'tsev operations.

Definition 4. An ternary operation $\varphi : A^3 \to A$ on a finite set A is called Mal'tsev if it satisfies the following identities

$$\varphi(x, y, y) = \varphi(y, y, x) = x, \quad \forall x, y \in A.$$

For instance, operation φ defined in Example 5 is Mal'tsev.

The following theorem has first been proved in [2].

Theorem 1. *Let φ be a Mal'tsev operation. Then* $\mathrm{CSP}(\mathrm{Inv}(\varphi))$ *is solvable in polynomial time.*

The proof given in [2] employs a sophisticated algorithm that heavily uses algebraic techniques and can hardly be understood without extensive knowledge of universal algebra. In this paper, we give a much shorter and simpler proof that is completely self-contained. Although the two algorithms have been designed mostly independently, they have a similar structure. To complete this section we outline the structure of the two algorithms and describe the main differences between them.

Both algorithms use *compact representations* of relations invariant under a Mal'tsev operation. Clearly, the size of an n-ary relation can be exponential in n. Therefore, if n is, for example, the number of variables in a CSP-instance, then to represent an n-ary relation we need something more sophisticated than just a list of tuples. One possible form of such representation is introduced in the next section; representations used in [2] are very similar, but have a much finer structure. Both representations exploit the *rectangularity* of relations invariant under a Mal'tsev operation (see the next section).

Given a CSP-instance $\mathcal{P} = (V, A, \{C_1, \ldots, C_k\})$, both algorithms progressively find compact representations of the sets of solutions of the instances $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k = \mathcal{P}$, where $\mathcal{P}_i = (V, A, \{C_1, \ldots, C_i\})$. In [2], this is done by constructing, in a certain complicated way, and then solving a system of linear equations such that a representation of $\mathcal{P}_{i+1}$ can be found as a basis of its solution space. Later, when working on [4], the second author observed that the same task can be fulfilled in a much more straightforward and simple way. This observation is in the core of the present paper.

3. Signatures and Representations

Let A be a finite set, let n be a positive integer, let $\mathbf{t} = (t_1, \ldots, t_n)$ be a n-ary tuple, and let $i_1, \ldots, i_j$ be (not necessarily different) elements from $[n]$. By $\mathrm{pr}_{i_1,\ldots,i_j} \mathbf{t}$ we denote the tuple $(t_{i_1}, \ldots, t_{i_j})$. Similarly, for every n-ary relation R on A and for every $i_1, \ldots, i_j \in [n]$, we denote by $\mathrm{pr}_{i_1,\ldots,i_j} R$ the j-ary relation given by $\{\mathrm{pr}_{i_1,\ldots,i_j} \mathbf{t} : \mathbf{t} \in R\}$.

Given a relation R and an operation φ, we denote by $\langle R \rangle_\varphi$ the smallest relation R' that contains R and that is invariant under φ. Very often, the operation φ will be clear from the context and we will drop it writing $\langle R \rangle$ instead of $\langle R \rangle_\varphi$.

Let n be a positive integer, let A be a finite set, let $\mathbf{t}$, $\mathbf{t'}$ be n-ary tuples and let (i, a, b) be any element in $[n] \times A^2$. We say that $(\mathbf{t}, \mathbf{t'})$ *witnesses* (i, a, b) if $\mathrm{pr}_{1,\ldots,i-1} \mathbf{t} = \mathrm{pr}_{1,\ldots,i-1} \mathbf{t'}$, $\mathrm{pr}_i \mathbf{t} = a$, and $\mathrm{pr}_i \mathbf{t'} = b$. We also say that $\mathbf{t}$ and $\mathbf{t'}$ witness (i, a, b) meaning that $(\mathbf{t}, \mathbf{t'})$ witnesses (i, a, b).

Let R be any n-ary relation on A. We define the *signature* of R, $\mathrm{Sig}_R \subseteq [n] \times A^2$, as the set containing all those $(i, a, b) \in [n] \times A^2$ witnessed by tuples in R, that is

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

A subset R' of R is called a *representation* of R if $\mathrm{Sig}_{R'} = \mathrm{Sig}_R$. If furthermore, $|R'| \leq 2|\,\mathrm{Sig}_R\,|$ then R' is called a *compact* representation of R. Observe that every relation R has compact representations. Indeed, in order to construct such a compact representation R' we only we only need to select, for each (i, a, b) in Sig_R a couple of tuples $\mathbf{t}, \mathbf{t'}$ in R that witness (i, a, b) and include them in R'.

Example 6. Fix a set A, an element $d \in A$, and an integer n. For every $(i, a) \in [n] \times A$ we define the tuple $\mathbf{e}^d_{i,a}$ as the only tuple satisfying

$$\mathrm{pr}_j \mathbf{e}^d_{i,a} = \begin{cases} a & \text{if } i = j \\ d & \text{otherwise} \end{cases} \quad \text{for all } j \in [n]$$

It is easy to observe that for every $(i, a, b) \in [n] \times A^2$, $(\mathbf{e}^d_{i,a}, \mathbf{e}^d_{i,b})$ witnesses (i, a, b). Consequently, for a fixed d, the set of tuples $\{\mathbf{e}^d_{i,a} : i \in [n], a \in A\}$ is a representation of the relation A^n. Notice also that it is indeed a compact representation.

The algorithm we propose relies on Lemma 1, which follows from the property of *rectangularity* of relations invariant under a Mal'tsev operation φ. Let R be an n-ary relation invariant under φ and $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3} \in R$ are such that $\mathrm{pr}_{1,\ldots,n-1} \mathbf{t_2} = \mathrm{pr}_{1,\ldots,n-1} \mathbf{t_3}$ and $\mathrm{pr}_n \mathbf{t_1} = \mathrm{pr}_n \mathbf{t_2}$. Then the tuple $\mathbf{t}$ with $\mathrm{pr}_{1,\ldots,n-1} \mathbf{t} = \mathrm{pr}_{1,\ldots,n-1} \mathbf{t_1}$ and $\mathrm{pr}_n \mathbf{t} = \mathrm{pr}_n \mathbf{t_3}$ belongs to R. Indeed, we can choose $\mathbf{t} = \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) \in R$. Let us see that $\mathbf{t}$ satisfies the required conditions. Since φ is Mal'tsev we can infer that

$$\mathrm{pr}_{1,\ldots,n-1} \mathbf{t} = \mathrm{pr}_{1,\ldots,n-1} \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) = \mathrm{pr}_{1,\ldots,n-1} \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_2}) = \mathrm{pr}_{1,\ldots,n-1} \mathbf{t_1}.$$

Also, we have that

$$\mathrm{pr}_n \mathbf{t} = \mathrm{pr}_n \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) = \mathrm{pr}_n \varphi(\mathbf{t_2}, \mathbf{t_2}, \mathbf{t_3}) = \mathrm{pr}_n \mathbf{t_3}.$$

Lemma 1. *Let A be a finite set, let $\varphi : A^3 \to A$ be a Mal'tsev operation, let R be a relation on A invariant under φ and let R' be a representation of R. Then $\langle R' \rangle_\varphi = R$*

Proof. Let n be the arity of R. By induction on i, we shall show that, for every $i \in [n]$, $\mathrm{pr}_{1,\dots,i}\langle R' \rangle = \mathrm{pr}_{1,\dots,i} R$. The case $i = 1$ follows easily from the fact that for each $\mathbf{t} \in R$, $(1, \mathrm{pr}_1 \mathbf{t}, \mathrm{pr}_1 \mathbf{t})$ is in Sig_R and hence in $\mathrm{Sig}_{R'}$.

So, let us assume that the claim holds for $i < n$ and let $\mathbf{t}$ be any tuple in R. We show that $\mathrm{pr}_{1,\dots,i+1} \mathbf{t} \in \mathrm{pr}_{1,\dots,i+1}\langle R' \rangle$. By induction hypothesis there exists a tuple $\mathbf{t_1}$ in $\langle R' \rangle$ such that $\mathrm{pr}_{1,\dots,i} \mathbf{t_1} = \mathrm{pr}_{1,\dots,i} \mathbf{t}$. We have that $(i+1, \mathrm{pr}_{i+1} \mathbf{t_1}, \mathrm{pr}_{i+1} \mathbf{t})$ belongs to Sig_R, and therefore, there exists some tuples $\mathbf{t_2}$ and $\mathbf{t_3}$ in R' witnessing it. To complete the proof we just need to observe that since $\mathbf{t_2}$ and $\mathbf{t_3}$ witness $(i + 1, \mathrm{pr}_{i+1} \mathbf{t}, \mathrm{pr}_{i+1} \mathbf{t_1})$ we have that $\mathrm{pr}_{1,\dots,i} \mathbf{t_2} = \mathrm{pr}_{1,\dots,i} \mathbf{t_3}$, and that $\mathrm{pr}_{i+1} \mathbf{t_1} = \mathrm{pr}_{i+1} \mathbf{t_2}$ and $\mathrm{pr}_{i+1} \mathbf{t} = \mathrm{pr}_{i+1} \mathbf{t_3}$. Then the equality $\mathrm{pr}_{1,\dots,i+1} \mathbf{t} = \mathrm{pr}_{1,\dots,i+1} \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})$ follows from the rectangularity of $\mathrm{pr}_{1,\dots,i+1} R$. ■

4. Proof of Theorem 1

We prove Theorem 1 by giving a polynomial-time algorithm that decides correctly whether a CSP(Inv(φ)) instance has a solution.

Let $\mathcal{P} = (\{v_1, \dots, v_n\}, A, \{C_1, \dots, C_m\})$ be a CSP(Inv(φ)) instance which will be the input of the algorithm.

For each $l \in \{0, \dots, m\}$ we define $\mathcal{P}_l$ as the CSP instance that contains the first l constraints of $\mathcal{P}$, that is $\mathcal{P}_l = (\{v_1, \dots, v_n\}, A, \{C_1, \dots, C_l\})$. Furthermore, we shall denote by R_l the n-ary relation on A defined as

$$R_l = \{(s(v_1), \dots, s(v_n)) : s \text{ is a solution of } \mathcal{P}_l)$$

As we have already mentioned, in a nutshell, the algorithm introduced in this section computes for each $l \in \{0, \dots, m\}$ a compact representation R'_l of R_l. In the initial case ($l = 0$), $\mathcal{P}_0$ does not have any constraint at all, and consequently, $R_0 = A^n$. Hence, a compact representation of R_0 can be easily obtained as in Example 6. Once a compact representation R'_0 of R_0 has been obtained then the algorithm starts an iterative process in which a compact representation R'_{l+1} of R_{l+1} is obtained from R'_l and the constraint C_{l+1}. This is achieved by calling procedure `Next`, which constitutes the core of the algorithm. The algorithm then, goes as follows:

Algorithm `Solve`$((\{v_1, \dots, v_n\}, A, \{C_1, \dots, C_m\}))$
Step 1 **select** an arbitrary element d in A
Step 2 **set** $R'_0 := \{\mathbf{e}^d_{i,a} : (i, a) \in [n] \times A\}$
Step 3 **for each** $l \in \{0, \dots, m-1\}$ **do**
(let C_{l+1} be $((v_{i_1}, \dots, v_{i_{l+1}}), S_{l+1}))$
Step 3.1 **set** $R'_{l+1} :=$ `Next`$(R'_l, i_1, \dots, i_{l+1}, S_{l+1})$
endfor
Step 4 **if** $R'_m \neq \emptyset$ **return yes**
Step 5 **otherwise return no**

Observe that if we modify step 4 so that the algorithm returns an arbitrary tuple in R'_m instead of "yes" then we have an algorithm that does not merely solve the decision question but actually provides a solution.

Correctness and polynomial time complexity of the algorithm is a direct consequence of the correctness and the running time of the procedure `Next`: As it is shown in Section 4.3 (Lemma 2) at each iteration of Step 3.1, the output of the call $\texttt{Next}(R'_l, i_1, \ldots, i_{l+1}, S_{l+1})$ is a compact representation of the relation $\{\mathbf{t} \in R_l : \mathrm{pr}_{i_1,\ldots,i_{l+1}} \mathbf{t} \in S_{l+1}\}$ which is indeed R_{l+1}. Furthermore the time complexity of `Solve` is obviously polynomial in the time complexity of `Next`. Thus to finish the proof of Theorem 1 we need to to design `Next` and show that it is correct and polynomial time. ■

The remaining part of the paper is devoted to defining and analyzing procedure `Next`. In order to define procedure `Next` it is convenient to introduce previously a pair of simple procedures, namely `Nonempty` and `Fix-values`, which will be intensively used by our procedure `Next`.

4.1. **Procedure `Nonempty`.** This procedure receives as an input a compact representation R' of a relation R invariant under φ, a sequence $i_1, \ldots, i_j$ of elements in $[n]$ where n is the arity of R, and a j-ary relation S also invariant under φ. The output of the procedure is either an n-ary tuple $\mathbf{t} \in R$ such that $\mathrm{pr}_{i_1,\ldots,i_j} \mathbf{t} \in S$ or "no" meaning that such a tuple does not exist.

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

We shall start by studying its correctness. First observe that every tuple in U belongs initially to R' (and hence to R), or it has been obtained by applying φ to some tuples $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}$ that previously belong to U. Therefore, since R is invariant under φ, we can conclude that $U \subseteq R$. Consequently, if a tuple $\mathbf{t}$ is returned in step 3, then it belongs to R and also satisfies the condition $\mathrm{pr}_{i_1,\ldots,i_j} \mathbf{t} \in S$, as desired. It only remains to show that if a "no" is returned in step 4 then there exists no tuple $\mathbf{t}$ in R such that $\mathrm{pr}_{i_1,\ldots,i_j} \mathbf{t} \in S$. In order to do this we need to show some simple facts about U. Notice that at any point of the execution of the procedure $R' \subseteq U$. Then U is also a representation of R and hence $\langle U \rangle = R$. Therefore we have that

$$\langle \mathrm{pr}_{i_1,\ldots,i_j} U \rangle = \mathrm{pr}_{i_1,\ldots,i_j} \langle U \rangle = \mathrm{pr}_{i_1,\ldots,i_j} R$$

By the condition on the "while" of step 2 we have that when the procedure leaves the execution of step 2 it must be case that for all $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3} \in U$, $\mathrm{pr}_{i_1,\ldots,i_j} \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) \in \mathrm{pr}_{i_1,\ldots,i_j} U$ and consequently $\mathrm{pr}_{i_1,\ldots,i_j} U = \langle \mathrm{pr}_{i_1,\ldots,i_j} U \rangle = \mathrm{pr}_{i_1,\ldots,i_j} R$. Hence, if there exists some $\mathbf{t}$ in R such that $\mathrm{pr}_{i_1,\ldots,i_j} \in S$ then it must exist some $\mathbf{t}'$ in U such that $\mathrm{pr}_{i_1,\ldots,i_j} \in S$ and we are done.

Let us study now the running time of the procedure. It is only necessary to focus on steps 2 and 3. At each iteration of the loop in step 2, cardinality of U increases by one. So we can bound the number of iterations by the size $|U|$ of U at the end of the execution of the procedure.

The cost of each of the iteration is basically dominated by the cost of checking whether there exists some tuples $\exists \mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3} \in U$ such that $\operatorname{pr}_{i_1,\dots,i_j} \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) \notin \operatorname{pr}_{i_1,\dots,i_j} U$ done in step 2. In order to try all possible combinations for $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}$ in U, $|U|^3$ steps suffice. Each one of these steps requires time $O(|U|n)$, as tuples have arity n and checking whether $\varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})$ belongs to U can be done naively by a sequential search in U. Thus, the running time of `Nonempty` is polynomial in n and in the possible size of U.

Notice now that this is not enough. Indeed, as we do not restrict the arity of the relations used, j may be comparable with n, which implies that the size of U may be exponential in n. To cope this problem, we organize invoking `Nonempty` in such a way that the size of U is bounded by a polynomial in $|S|$, n and $|A|$. In order to accomplish this we shall ensure that at each call to `Nonempty` the following condition is satisfied

$$|\operatorname{pr}_{i_1,\dots,i_j} R| \leq |S| \cdot |A|^2 \tag{1}$$

Later (see procedures `Next` and `Next-beta`) we show how this can be achieved. It is not difficult to see that if (1) is true then the size of U can be bounded by a polynomial. More precisely, the size of U can be bounded by the initial size of R' which is at most $n \cdot |A|^2$ (since R' is compact) plus the number of iterations of step 2, which is bounded by $|\operatorname{pr}_{i_1,\dots,i_j} R| \leq |S| \cdot |A|^2$.

4.2. **Procedure `Fix-values`.** This procedure receives as an input a compact representation R' of a relation R invariant under φ and a sequence $a_1, \dots, a_m$, $m \leq n$ of elements of A (n is the arity of R). The output is a compact representation of the relation given by

$$\{\mathbf{t} \in R : \operatorname{pr}_1 \mathbf{t} = a_1, \dots, \operatorname{pr}_m \mathbf{t} = a_m\}$$

Procedure `Fix-values`$(R', a_1, \dots, a_m)$
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)
(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 $U_{j+1} := U_{j+1} \cup \{\mathbf{t_1}, \varphi(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})\}$
endfor
Step 2.4 **set** j:=j+1
endwhile
Step 3 **return** U_m

Let us study the correctness of the procedure. We shall show by induction on $j \in \{0, \dots, m\}$ that U_j is a compact representation of $R_j = \{\mathbf{t} \in R : \operatorname{pr}_1 \mathbf{t} = a_1, \dots, \operatorname{pr}_j \mathbf{t} = a_j\}$. The case $j = 0$ is correctly settled in step 1. Hence it is only necessary to show that at every iteration of the **while** loop in step 2, if U_j is a compact representation of R_j then U_{j+1} is a compact representation of R_{j+1}. It is easy to see that if any of the conditions of the **if** in step 2.2.1 is falsified then (i,a,b) is not in $\mathrm{Sig}_{R_{j+1}}$. So it only remains to see that when the **if** in step 2.2.1 is satisfied, we have that (a) $\mathbf{t_1}$ and $\varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3})$ are tuples in R_{j+1}, and (b) $(\mathbf{t_1},\varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3}))$ witnesses (i,a,b).

Proof of (a): As $\mathbf{t_1} = \mathtt{Nonempty}(U_j, j+1, i, \{(a_{j+1},a)\})$, we can conclude that $\mathbf{t_1}$ belongs to R_j, $\mathrm{pr}_{j+1}\,\mathbf{t_1} = a_{j+1}$, and $\mathrm{pr}_i\,\mathbf{t_1} = a$. Consequently $\mathbf{t_1}$ belongs to R_{j+1}. Furthermore, as $\mathbf{t_1}$, $\mathbf{t_2}$, and $\mathbf{t_3}$ are in R_j and R_j is invariant under φ, $\varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3})$ belongs to R_j. Let us see now that $\mathrm{pr}_{j+1}\,\mathbf{t_2} = \mathrm{pr}_{j+1}\,\mathbf{t_3}$ by means of a case analysis. If $i > j+1$ then we have that $\mathrm{pr}_{j+1}\,\mathbf{t_2} = \mathrm{pr}_{j+1}\,\mathbf{t_3}$ as $(\mathbf{t_2},\mathbf{t_3})$ witnesses (i,a,b). If $i \le j+1$ then $a = b = a_i$ and hence $\mathbf{t_2}$ and $\mathbf{t_3}$ are identical.

Finally, since φ is Mal'tsev, $\mathrm{pr}_{j+1}\,\varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3}) = \mathrm{pr}_{j+1}\,\mathbf{t_1} = a_{j+1}$ and hence $\varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3})$ belongs to R_{j+1}.

Proof of (b): Since $(\mathbf{t_2},\mathbf{t_3})$ witnesses (i,a,b) we have that $\mathrm{pr}_{1,\dots,i-1}\,\mathbf{t_2} = \mathrm{pr}_{1,\dots,i-1}\,\mathbf{t_3}$ Consequently, $\mathrm{pr}_{1,\dots,i-1}\,\varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3}) = \mathrm{pr}_{1,\dots,i-1}\,\mathbf{t_1}$. Furthermore we also have that $\mathrm{pr}_i\,\varphi(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3}) = \mathrm{pr}_i\,\varphi(a,a,b) = b$.

Notice that each iteration adds at most 2 tuples for some (i,a,b) in $\mathrm{Sig}_{R_{j+1}}$. Consequently, U_{j+1} is compact. This completes the proof of its correctness.

Let us study now the time complexity of `Fix-values`. The **while** loop at step 2 is performed $m \le n$ times. At each iteration the procedure executes another loop (step 2.2). The **for each** loop at step 2.2 is executed for each (i,a,b) in $[n]\times A^2$, that is, a total number of $n|A|^2$ times. The cost of each iteration of the loop is basically dominated by the cost of the call to procedure `Nonempty`. Therefore, the time complexity of the procedure is polynomial in n and the maximal time complexity of `Nonempty`. Observe that in this case condition (1) in the call of `Nonempty` is satisfied.

4.3. **Procedure `Next`.** We are now almost in a position to introduce procedure `Next`. Procedure `Next` receives as input a compact representation R' of a relation R invariant under φ, a sequence $i_1,\dots,i_j$ of elements in $[n]$ where n is the arity of R, and a j-ary relation S invariant under φ. The output of `Next` is a compact representation of the relation $R^* = \{\mathbf{t} \in R : \mathrm{pr}_{i_1,\dots,i_j}\,\mathbf{t} \in S\}$. It is an easy exercise to verify that R^* is also invariant under φ.

We shall start by defining a procedure, called `Next-beta` that is equivalent to `Next`, but has a worse running time. In particular, the running time of `Next-beta` might be exponential in the size of its input.

Procedure `Next-beta`$(R', i_1,\dots,i_j, S)$
Step 1 **set** $U := \emptyset$
Step 2 **for each** $(i,a,b) \in [n]\times A^2$ **do**
Step 2.1 **if** `Nonempty`$(R', i_1,\dots,i_j, i, S\times\{a\})\} \neq$"no" **then**
(let $\mathbf{t}$ be the tuple returned by `Nonempty`$(R', i_1,\dots,i_j, i, S\times\{a\})\})$
Step 2.2 **if** `Nonempty(Fix-values`$(R', \mathrm{pr}_1\,\mathbf{t},\dots,\mathrm{pr}_{i-1}\,\mathbf{t})$,
$i_1,\dots,i_j, i, S\times\{b\}) \neq$"no"
(let $\mathbf{t}'$ be the tuple returned by
`Nonempty(Fix-values`$(R', \mathrm{pr}_1\,\mathbf{t},\dots,\mathrm{pr}_{i-1}\,\mathbf{t}), i_1,\dots,i_j, i, S\times\{b\}))$
set $U := U \cup \{\mathbf{t},\mathbf{t}'\}$
endfor
Step 3 **return** U

The overall structure of procedure `Next-beta` is similar to that of procedure `Fix-values`. Observe that the condition of the **if** statement

$$\texttt{Nonempty}(R', i_1, \ldots, i_j, i, S \times \{a\})\} \neq \text{"no"}$$

of step 2.1 is satisfied if and only if there exists a tuple $\mathbf{t} \in R$ such that $\mathrm{pr}_{i_1,\ldots i_j} \mathbf{t} \in S$ and $\mathrm{pr}_i \mathbf{t} = a$. Hence if such a tuple does not exist then (i, a, b) is not in Sig_{R^*} and nothing needs to be done for (i, a, b). Now consider the condition of the **if** statement in step 2.2 which is given by

$$\texttt{Nonempty}(\texttt{Fix-values}(R', \mathrm{pr}_1 \mathbf{t}, \ldots, \mathrm{pr}_{i-1} \mathbf{t}), i_1, \ldots, i_j, i, S \times \{b\}) \neq \text{"no"}$$

This condition is satisfied if and only if there exists some $\mathbf{t}'$ in R such that $\mathrm{pr}_{i_1,\ldots,i_j} \mathbf{t}' \in S$ such that $\mathrm{pr}_{1,\ldots,i-1} \mathbf{t}' = \mathrm{pr}_{1,\ldots,i-1} \mathbf{t}$ and $\mathrm{pr}_i \mathbf{t}' = b$. It is immediate to see that if the condition holds then $(\mathbf{t}, \mathbf{t}')$ witnesses (i, a, b). It only remains to show that if $(i, a, b) \in \mathrm{Sig}_{R^*}$ then such a $\mathbf{t}'$ must exist. In order to do this, it is only necessary to verify that if $\mathbf{t_a}, \mathbf{t_b}$ are tuples in R^* witnessing (i, a, b) then, since φ is Mal'tsev, the tuple $\varphi(\mathbf{t}, \mathbf{t_a}, \mathbf{t_b})$ satisfies the desired properties (here $\mathbf{t}$ is the tuple returned by the call to procedure `Nonempty` in step 2.1).

Again, the cardinality of U is bounded by $2|\mathrm{Sig}_{R^*}|$ and, hence, U is a compact representation.

Let us study the running time of procedure `Next-beta`. The loop of step 2 is performed $n|A|^2$ times and the cost of each iteration is basically the cost of steps 2.1 and 2.2 in which other procedures are called. Therefore, the running time of `Next-beta` is polynomial in n and the running time of `Nonempty` and `Fix-values`. However, when invoking `Nonempty` on steps 2.1 and 2.2, we cannot be sure that condition (1) holds, and therefore the running time of `Nonempty` and consequently `Next-beta` may be exponential in the size of the input. To avoid this problem, we define a new procedure `Next` which makes a sequence of calls to `Next-beta` such that the following condition is satisfied:

$$|\mathrm{pr}_{i_1,\ldots,i_j} R| \leq |S| \cdot |A| \tag{2}$$

It is easy to observe that if condition (2) in the call of `Next-beta` is guaranteed then condition (1) in every call of `Nonempty` is also satisfied. Consequently, if `Next-beta` is called with parameters satisfying condition (2) then its running time is polynomial on n, S and $|A|$.

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`$(U_l, i_1, \ldots, i_{l+1}, \mathrm{pr}_{i_1,\ldots,i_{l+1}} S)$
end while
Step 3 **return** U_j

Let us show that condition (2) is satisfied in any call to Procedure `Next-beta` in step 2.1. In the first iteration ($l = 0$), condition (2) holds as $\mathrm{pr}_{i_1} R' \subseteq A$. For every subsequent iteration $l \in \{1, \ldots, j-1\}$ observe that at the beginning of the iteration,

$$\mathrm{pr}_{i_1,\ldots,i_l} \langle U_l \rangle = \mathrm{pr}_{i_1,\ldots,i_l} S$$

Consequently, at each call to Procedure `Next-beta` we have

$$|\mathrm{pr}_{i_1,\ldots,i_{l+1}} \langle U_l \rangle| \leq |\mathrm{pr}_{i_1,\ldots,i_l} S||A| \leq |S| \cdot |A|.$$

Therefore, condition (2) holds and the running time of the call is polynomial in n, $|S|$ and $|A|$.

Therefore, we have just proved

Lemma 2. *For every $n \geq 1$, every n-ary relation R invariant under φ, every compact representation R' of R, every $i_1, \ldots, i_j \in [n]$, and every j-ary relation S invariant under φ,* $\texttt{Next}(R', i_1, \ldots, i_j, S)$ *computes a compact representation of $R^* = \{\mathbf{t} \in R : \mathrm{pr}_{i_1,\ldots,i_j} \in S\}$ in a time polynomial in n, $|S|$ and $|A|$. Furthermore R^* is invariant under φ.*

Corollary 1. *Algorithm* `Solve` *decides correctly if an instance $\mathcal{P}$ of* $\mathrm{CSP}(\mathrm{Inv}(\varphi))$ *is satisfiable in a time polynomial in n, s, and $|A|$, where n is the number of variables of $\mathcal{P}$, s is the total number of tuples in the constraint relations, and A is the domain.*

References

[1] A. Bulatov. A Dichotomy Theorem for Constraints on a Three-element Set. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, (FOCS'02)*, pages 649–658, 2002.

[2] A. Bulatov. Mal'tsev Constraints are Tractable. Technical Report PRG-02-05, Computing Laboratory, Oxford University, 2002.

[3] A. Bulatov. Tractable Conservative Constraint Satisfaction Problems. In *Proceedings of the 18th Annual IEEE Simposium on Logic in Computer Science, (LICS'03)*, pages 321–330, 2003.

[4] A. Bulatov, H. Chen, and V. Dalmau. Learnability of Relatively Quantified Generalized Formulas. In *Proceedings of 15th International Conference on Algorithmic Learning Theory, (ALT '04)*, number 3244 in Lecture Notes in Computer Science, pages 365–379, 2004.

[5] A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *Proceedings 44rd IEEE Symposium on Foundations of Computer Science, FOCS'03*, pages 562–571, Boston, USA, October 2003.

[6] A.A. Bulatov and P.G. Jeavons. Algebraic structures in combinatorial problems. Technical Report MATH-AL-4-2001, Technische universität Dresden, Dresden, Germany, 2001.

[7] A.A. Bulatov, A.A. Krokhin, and P.G. Jeavons. Classifying complexity of constraints using finite algebras. *SIAM J. Computing*, 34(3):720–742, 2005.

[8] A.A. Bulatov, A.A. Krokhin, and P.G. Jeavons. Constraint satisfaction problems and finite algebras. In *Proceedings of 27th International Colloquium on Automata, Languages and Programming—ICALP'00*, volume 1853 of *Lecture Notes in Computer Science*, pages 272–282. Springer-Verlag, 2000.

[9] S.A. Cook. The Complexity of Theorem-Proving Procedures. In *3rd Annual ACM Symposium on Theory of Computing, (STOC'71)*, pages 151–158, 1971.

[10] V. Dalmau. A New Tractable Class of Constraint Satisfaction Problems. *Ann. Math. Artif. Intell.*, 44(1–2):61–85, 2005.

[11] V. Dalmau and J. Pearson. Set Functions and Width 1. In *5th International Conference on Principles and Practice of Constraint Programming, (CP'99)*, volume 1713 of *Lecture Notes in Computer Science*, pages 159–173, Berlin/New York, 1999. Springer-Verlag.

[12] T. Feder. Constraint satisfaction on finite groups with near subgroups. Electronic Colloquium on Computational Complexity (ECCC) TR05-005, 2005.

[13] T. Feder and M.Y. Vardi. The Computational Structure of Monotone Monadic SNP and Contraint Satisfaction: A Study through Datalog and Group Theory. *SIAM J. Computing*, 28(1):57–104, 1998.

[14] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H Freeman and Company, 1979.

[15] P. Jeavons, D. Cohen, and M. Gyssens. Closure Properties of Constraints. *Journal of the ACM*, 44(4):527–548, July 1997.

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

[17] P.G. Jeavons, D.A. Cohen, and M.C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1-2):251–265, 1998.

[18] A. Krokhin, A. Bulatov, and P. Jeavons. The complexity of constraint satisfaction: An algebraic approach. In *Proceedings of SMS-NATO Meeting on Structural Theory of Automata, Semigroups and Universal Algebra*, Montreal, Canada, 2003. to appear.

[19] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[20] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[21] T.J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, (STOC'78)*, pages 216–226, 1978.