

GENERALIZED MAJORITY-MINORITY OPERATIONS ARE TRACTABLE

VÍCTOR DALMAU

Departament de Tecnologia, Universitat Pompeu Fabra,, Estació de França, Passeig de la Circum-
val.lacio 8. Barcelona 08003, Spain
e-mail address: victor.dalmau@tecn.upf.es

ABSTRACT. Generalized majority-minority (GMM) operations are introduced as a common generalization of near unanimity operations and Mal'tsev operations on finite sets. We show that every instance of the constraint satisfaction problem (CSP), where all constraint relations are invariant under a (fixed) GMM operation, is solvable in polynomial time. This constitutes one of the largest tractable cases of the CSP.

1. INTRODUCTION

Constraint satisfaction problems arise in a wide variety of domains, 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 domain, and a set of constraints, where a constraint is a pair given by a list of variables, called scope, and a relation indicating the valid combinations of values for the variables in the scope; the goal is to decide whether or not there is an assignment of values to the variables satisfying all of the constraints. It is well known that the CSP admits several different but equivalent definitions. Feder and Vardi [19] formulated it as the problem of deciding whether there exists an homomorphism between two given relational structures. Also, an instance of the CSP can be viewed as a positive primitive sentence; the question is to decide whether or not the sentence is true.

In its full generality the CSP is NP-complete. This fact motivates the project of identifying restricted subclasses of the problem that are solvable in polynomial time. The most customary way to restrict the CSP is by fixing a set of relations Γ , generally called *constraint language* or *basis* and consider only instances of the CSP in which every relation appearing in a constraint belongs to Γ ; this restricted version of the problem is generally denoted by $\text{CSP}(\Gamma)$. Much effort has been devoted to the goal of isolating those constraint languages, Γ , for which its associated constraint satisfaction problem, $\text{CSP}(\Gamma)$, is polynomial-time solvable. Despite the large amount of results in this direction [1, 2, 3, 4, 5, 8, 9, 10, 11, 14, 16, 17, 18, 19, 22, 23, 24], a complete classification is still not known.

2000 ACM Subject Classification: F.4.1.

Key words and phrases: constraint satisfaction, complexity, Mal'tsev, near-unanimity.

The other usual way to define subclasses of the general CSP is by restricting the possible scopes, not the relations, that can appear in a constraint. The state of affairs here is a way better: It has been proved [20] that, under certain plausible assumptions, the tractable class identified in [15] is the only one, settling completely the question.

Our goal in this paper is to introduce a general condition, such that every constraint language Γ satisfying this condition, leads to a subclass of the CSP, $\text{CSP}(\Gamma)$, solvable in polynomial time. In order to place our result in context it will be necessary to include a short description of the cases of the CSP, known to be tractable.

In our classification we shall distinguish between *pure* and *hybrid* tractable cases. Intuitively, *pure* tractable cases are those that are explained by a simple and concrete combinatorial principle whereas *hybrid* tractable cases are those that can be reduced (not necessarily without a considerable degree of sophistication) to a combination of pure cases. Let us recall here that this distinction is our rather personal attempt to classify the tractable cases of the CSP and does not pretend to be any claim about the “true nature” of the tractable cases.

According to our view there are basically three pure maximal tractable cases: *width 1*, *bounded strict width*, and *Mal'tsev* problems. This classification corresponds to the three tractable families isolated in Feder and Vardi [19]: width 1, bounded strict width, and subgroup problems, in which the latter has been enlarged as to include all Mal'tsev problems, proven to be tractable recently by Bulatov [2] (see also [7]). The vast majority of tractable cases of the CSP identified in the past (prior [10]) fall in one of these three pure categories. As an illustrative example consider the six tractable cases in the boolean domain identified by Schaefer [26], namely, 0-valid, 1-valid, Horn, dual-Horn, bijunctive and affine problems. The first four classes are particular instances of width 1 problems whereas the fifth and sixth class belong to bounded strict width and Mal'tsev respectively.

In [10], the pursuit of new tractable cases of the CSP took a new direction. The new class identified in [10], the so-called paper-scissor-stone problems, could be regarded as constituted by an amalgam of Horn problems (and hence width 1 problems) and dual discriminator problems [13], known to be particular instances of bounded strict width problems. The algorithm devised in [10] exploits the fact that the interaction between the width 1 part and the bounded strict width part of the instance is very constrained. The class of paper-scissor-stone problems is the first hybrid tractable case.

All tractable cases identified after [10] with the notable exception of Mal'tsev problems are hybrid problems (the references [1, 3, 4, 5, 11, 16] constitute, up to the best of our knowledge, a complete list). Indeed, it is not daring to say that is very likely that much of the future progress in the study of the complexity of the CSP will come from a better understanding of the interaction between the different sources of pure tractability.

A unifying framework for tractability of constraint satisfaction has been developed by Jeavons and coauthors in a sequence of papers culminating in [23]; the key theme of this framework is that it is generally possible to explain the tractability of a certain subclass of the CSP, $\text{CSP}(\Gamma)$, by means of certain algebraic invariance properties of the relations in Γ . Consider, for instance, the class of bounded strict width problems: It is well known [19] (see also [22]) that a constraint language Γ is bounded strict width if and only if there exists a near-unanimity operation φ , namely, an operation $\varphi : A^k \rightarrow A$ with $k \geq 3$ satisfying

$$\varphi(x, y, \dots, y) = \varphi(y, x, \dots, y) = \dots = \varphi(y, y, \dots, x) = y$$

for all $x, y \in A$, such that every relation in Γ is invariant under φ .

In a similar vein, a constraint language Γ is Mal'tsev if all its relations are invariant under a Mal'tsev operation, ie, an operation $\varphi : A^3 \rightarrow A$ satisfying

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

Similar characterizations, in terms of algebraic invariance properties, are known for the vast majority of tractable cases of the CSP. In fact, the connection between tractability of CSP and invariance properties is tighter, as it can be shown that the complexity of a given subclass of the CSP, $\text{CSP}(\Gamma)$, depends *only* on the the set of operations under which Γ is invariant [21].

Generalized majority-minority operations first arose in the study of the learnability of relatively quantified generalized formulas [6]. An operation $\varphi : A^k \rightarrow A$ with $k \geq 3$ is a *generalized majority-minority (GMM) operation* if for all $a, b \in A$,

$$\varphi(x, y, \dots, y) = \varphi(y, x, \dots, y) = \dots = \varphi(y, y, \dots, x) = y$$

for all $x, y \in \{a, b\}$

or

$$\varphi(x, y, \dots, y) = \varphi(y, y, \dots, x) = x \text{ for all } x, y \in \{a, b\}.$$

GMM operations generalize both near-unanimity and Mal'tsev operations. Intuitively, an operation is GMM if for each 2-element subset of its domain acts either as a near-unanimity or as a Mal'tsev. In this paper we prove that every constraint language Γ in which all its relations are invariant under a GMM operation φ , gives rise to a subclass of the CSP solvable in polynomial-time. This new family of constraint satisfaction problems includes among other all bounded strict width and all Mal'tsev problems and, hence, constitutes one of the largest tractable classes of the CSP. Our algorithm exploits a feature which is already used -at least implicitly- in the known algorithms for Mal'tsev and Near-unanimity problems: for any relation R invariant under a GMM operation φ it is always possible to obtain a "succinct" representation, in the form of a relation $G \subseteq R$ that *generates* R , i.e., such that R is the smallest relation invariant under φ containing G . Indeed, our algorithm for GMM problems can be viewed as a mixture of the algorithms for Near-unanimity and Mal'tsev problems. However, it should be point out that the interaction between the two conditions is rather intricate.

2. PRELIMINAIRES

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 $\{1, \dots, n\}$.

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

Definition 2.1. An instance of a *constraint satisfaction problem* consists of:

- a finite set of variables, $V = \{v_1, \dots, v_n\}$;
- a finite domain of values, A ;
- a finite set of constraints $\{C_1, \dots, C_m\}$; each constraint C_l , $l \in [m]$, is a pair $((v_{i_1}, \dots, v_{i_{k_l}}), S_l)$ where:
 - $(v_{i_1}, \dots, v_{i_{k_l}})$ 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 *constraint relation*.

A *solution* to a constraint satisfaction instance is a mapping $s : V \rightarrow A$ such that for each constraint C_l , $l \in [m]$, we have that $(s(v_{i_1}), \dots, s(v_{k_l}))$ belongs to S_l . Deciding whether or not a given problem instance has a solution is NP-complete in general, even when the constraints are restricted to binary constraints [25] or the domain of the problem has size 2 [12]. However by imposing restrictions on the constraint relations it is possible to obtain restricted versions of the problem that are tractable.

Definition 2.2. For any set of relations Γ , $\text{CSP}(\Gamma)$ 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?

In the last few years much effort has been devoted to the identification of those sets Γ for which $\text{CSP}(\Gamma)$ is solvable in polynomial time (See [1, 2, 3, 4, 5, 8, 9, 10, 11, 14, 16, 17, 18, 19, 22, 23, 24]). In order to isolate such “islands of tractability” it has been particularly useful to consider certain closure conditions on the relations of Γ . In order to make this more precise we need to introduce the following definition, which constitutes the cornerstone of the so-called *algebraic approach* of the study of the CSP.

Definition 2.3. Let $\varphi : A^k \rightarrow A$ be an k -ary operation on A and let R be a n -ary relation over A . We say that R is invariant under φ if for all (not necessarily different) tuples $\mathbf{t}_1 = (t_1^1, \dots, t_n^1), \dots, \mathbf{t}_k = (t_1^k, \dots, t_n^k)$ in R , the tuple $\varphi(\mathbf{t}_1, \dots, \mathbf{t}_k)$ defined as

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

belongs to R .

Given a relation R and an operation φ , we denote by $\langle R \rangle_\varphi$ the smallest relation S that contains R and that it 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 $\varphi : A^k \rightarrow A$ be any operation on A . We denote by $\text{Inv}(\varphi)$ the set containing all relations on A invariant under φ .

The vast majority of constraint languages Γ such that $\text{CSP}(\Gamma)$ is in PTIME can be expressed as $\text{Inv}(\varphi)$ for some operation φ . We refer the reader to the references pointed out in the introduction for a complete (up to the best of our knowledge) list of operations that lead to a tractable class of the CSP. In what follows we shall introduce only two families of operations, namely near-unanimity and Malt’sev, which will be particularly relevant to our work.

Example 2.4. (Near-Unanimity Operations) An operation $\varphi : A^k \rightarrow A$ with $k \geq 3$ is near-unanimity (NU) if for all $x, y \in A$, we have that

$$\varphi(x, y, \dots, y) = \varphi(y, x, \dots, y) = \dots = \varphi(y, y, \dots, x) = y$$

Tractability of $\text{CSP}(\text{Inv}(\varphi))$ for any arbitrary near-unanimity operation φ was proved in [19] (See also [22]).

Many well known tractable cases of the CSP, such as 2-SAT, or the family of CSP with implicative constraints [24, 13] are, in fact, particular instances of this general case.

Example 2.5. (Mal’tsev Operations) An operation $\varphi : A^3 \rightarrow A$ is Mal’tsev if for all $x, y \in A$, we have

$$\varphi(x, y, y) = \varphi(y, y, x) = x$$

In [2] (see [7] for a simpler proof), it was shown that for every Malt'sev operation φ , $\text{CSP}(\text{Inv}(\varphi))$ is solvable in polynomial time. This general result encompasses some previously known tractable cases of the CSP, such as CSP with constraints defined by a system of linear equations [23] or CSP with near-subgroups and its cosets [19, 18].

The class of generalized majority-minority operations generalizes both near-unanimity and Mal'tsev operations.

Definition 2.6. An operation $\varphi : A^k \rightarrow A$ with $k \geq 3$ is a *generalized majority-minority (GMM) operation* if for all $a, b \in A$, either

$$\varphi(x, y, \dots, y) = \varphi(y, x, \dots, y) = \dots = \varphi(y, y, \dots, x) = y \quad (2.1)$$

for all $x, y \in \{a, b\}$

or

$$\varphi(x, y, \dots, y) = \varphi(y, y, \dots, x) = x \quad \text{for all } x, y \in \{a, b\} \quad (2.2)$$

Generalized majority-minority operations were introduced in the study of the learnability of relatively quantified generalized formulas [6].

Let us fix a GMM operation on a set A . A pair $a, b \in A$ is said to be a *majority* pair if φ on a, b satisfies (2.1). It is said to be a *minority* pair if φ satisfies (2.2). If $a = b$ then we will say $\{a, b\}$ is a majority pair.

In this paper we prove the following result:

Theorem 2.7. *For every GMM operation φ , $\text{CSP}(\text{Inv}(\varphi))$ is solvable in polynomial time.*

The proof is given in Section 4.

3. SIGNATURES AND REPRESENTATIONS

Let A be a finite set, let n be a positive integer, let $\mathbf{t} = (t_1, \dots, t_n)$ be a n -ary tuple of elements in A , and let i_1, \dots, i_j elements in $[n]$. By $\text{pr}_{i_1, \dots, i_j} \mathbf{t}$ we denote the tuple $(t_{i_1}, \dots, t_{i_j})$. Similarly, for every n -ary relation R on A and for every $i_1, \dots, i_j \in [n]$ we denote by $\text{pr}_{i_1, \dots, i_j} R$ the j -ary relation given by $\{\text{pr}_{i_1, \dots, i_j} \mathbf{t} : \mathbf{t} \in R\}$. Given a subset $I = \{i_1, \dots, i_j\}$ of $[n]$ with $i_1 < i_2 < \dots < i_j$ we shall use $\text{pr}_I R$ to denote $\text{pr}_{i_1, \dots, i_j} R$.

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 $[n] \times A^2$ with $a \neq b$. We say that $(\mathbf{t}, \mathbf{t}')$ witnesses (i, a, b) if $\text{pr}_{1, \dots, i-1} \mathbf{t} = \text{pr}_{1, \dots, i-1} \mathbf{t}'$, $\text{pr}_i \mathbf{t} = a$, and $\text{pr}_i \mathbf{t}' = b$. We also say that \mathbf{t} and \mathbf{t}' witness (i, a, b) meaning that $(\mathbf{t}, \mathbf{t}')$ witnesses (i, a, b) .

Let $\varphi : A^k \rightarrow A$, $k \geq 3$, be a GMM operation and let R be any n -ary relation on A (not necessarily invariant under φ). We define the *signature* of R relative to φ , $\text{Sig}_R \subseteq [n] \times A^2$, as the set containing all those $(i, a, b) \in [n] \times A^2$, with $\{a, b\}$ a minority pair witnessed by tuples in R , that is

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

Fix a non-negative integer j . A subset R' of R is called a *representation* of R relative to φ of order j if $\text{Sig}_R = \text{Sig}_{R'}$ and for every $I \subseteq \{1, \dots, n\}$ with $|I| \leq j$, $\text{pr}_I R = \text{pr}_I R'$. We also say that R' is a j -representation of R relative to φ . Observe that for any n -ary relation there exists a j -representation with size bounded above by $2|\text{Sig}_R| + \sum_{I \subseteq [n], |I| \leq j} |\text{pr}_I R|$. We call any such representation, a *compact representation* of R . Note: When the operation

φ is clear from the context we shall drop the “relative to φ ” with the implicit understanding that signatures and representations are relative to φ .

Example 3.1. Let A be a finite set, let $\varphi : A^k \rightarrow A$, $k \geq 3$, be a GMM operation and let j and n positive integers. We shall construct a j -representation R' of $R = A^n$.

Initially R' is empty. Fix any arbitrary element d in A . First, observe that Sig_R contains all (i, a, b) in $[n] \times A^2$ where $\{a, b\}$ is a minority pair. For each triple (i, a, b) in Sig_R we add to R' two tuples $\mathbf{t}_a^i, \mathbf{t}_b^i$, where \mathbf{t}_a^i is the tuple that has a in its i th coordinate and d elsewhere and, accordingly, \mathbf{t}_b^i is the tuple that has b in its i th coordinate and d elsewhere. Notice that $(\mathbf{t}_a^i, \mathbf{t}_b^i)$ witnesses (i, a, b) . Hence, after adding to R' a corresponding pair $\mathbf{t}_a^i, \mathbf{t}_b^i$ for each (i, a, b) in Sig_R we have that $\text{Sig}_R = \text{Sig}_{R'}$. In a second step we add for each $i_1, \dots, i_{j'}$ with $j' \leq j$ and $i_1 < i_2 < \dots < i_{j'}$, and every $a_1, \dots, a_{j'} \in A$, the tuple $\mathbf{t}_{a_1, \dots, a_{j'}}^{i_1, \dots, i_{j'}}$ that has a_l in its i_l th coordinate for each $l \in [j']$, and d elsewhere. It is easy to verify that we obtain a relation R' such that for all $I \subseteq [n]$ with $|I| \leq j$, $\text{pr}_I R' = A^{|I|} = \text{pr}_I R$. Hence R' is a representation of R of order j . Observe that, indeed, R' is a *compact representation* of R .

The algorithm we propose relies on the following lemma.

Lemma 3.2. *Let A be a finite set, let $\varphi : A^k \rightarrow A$, $k \geq 3$ be a GMM operation, let R be a relation on A invariant under φ and let R' be a representation of R of order $k - 1$. Then $\langle R' \rangle = R$*

Proof. Let n be the arity of R . We shall show that for every $i \in [n]$, $\text{pr}_{1, \dots, i} \langle R' \rangle = \text{pr}_{1, \dots, i} R$ by induction on i . The case $i \leq k - 1$ follows from $\text{pr}_{1, \dots, k-1} R' = \text{pr}_{1, \dots, k-1} R$. So let $i \geq k$ and let $\mathbf{a} = (a_1, \dots, a_i) \in \text{pr}_{1, \dots, i} R$. By induction hypothesis, for some b_i , the tuple $\mathbf{a}' = (a_1, \dots, a_{i-1}, b_i)$ belongs to $\text{pr}_{1, \dots, i} \langle R' \rangle$. In what follows we shall denote $\text{pr}_{1, \dots, i} \langle R' \rangle$ as S .

We consider two cases.

Case 1. $\{a_i, b_i\}$ is majority.

In this case we show that, for every $I \subseteq \{1, \dots, i\}$, $\text{pr}_I \mathbf{a} \in \text{pr}_I S$. We show it by induction on the cardinality m of I . The result is true for $m \leq k - 1$ due to the fact that R' is a $(k - 1)$ -representation of R . It is also true for every set I that does not contain i , since $\mathbf{a}' \in S$ certifies it. Thus let $I = \{j_1, \dots, j_m\}$ be any set of indices $1 \leq j_1 < j_2 < \dots < j_m = i$ with $m \geq k$ and also let $\mathbf{a}^* = \text{pr}_I \mathbf{a}$. To simplify the notation let us denote $\mathbf{a}^* = (c_1, \dots, c_m)$. By induction hypothesis, $\text{pr}_I S$ contains the tuples $\mathbf{d}_1 = (d_1, c_2, \dots, c_m)$, $\mathbf{d}_2 = (c_1, d_2, c_3, \dots, c_m)$, \dots , $\mathbf{d}_m = (c_1, \dots, c_{m-1}, d_m)$ for some $d_1, \dots, d_m \in A$. If for some i , $c_i = d_i$ then we are done. Otherwise, we can assume that $d_m = b_i$ and $c_m = a_i$ and henceforth $\{d_m, c_m\}$ is majority. If for some j , the pair $\{d_j, c_j\}$ is minority then we are done, because $\varphi(\mathbf{d}_j, \mathbf{d}_j, \dots, \mathbf{d}_j, \mathbf{d}_m) = \mathbf{a}^*$. Otherwise, $\{d_j, c_j\}$ is majority for any j . In this case we have $\varphi(\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_{k-1}, \mathbf{d}_k) = \mathbf{a}^*$.

Case 2. $\{a_i, b_i\}$ is minority.

Since \mathbf{a} and \mathbf{a}' belong to $\text{pr}_{1, \dots, i} R$ then there exists some tuples $\mathbf{t}, \mathbf{t}' \in R$ such that $\text{pr}_{1, \dots, i} \mathbf{t} = \mathbf{a}$ and $\text{pr}_{1, \dots, i} \mathbf{t}' = \mathbf{a}'$. Consequently, \mathbf{t} and \mathbf{t}' witness (i, a_i, b_i) and since $\text{Sig}_{R'} = \text{Sig}_R$ we can conclude that $\text{pr}_{1, \dots, i} R'$ (and hence S) contains tuples $\mathbf{c} = (c_1, \dots, c_{i-1}, a_i)$ and $\mathbf{c}' = (c_1, \dots, c_{i-1}, b_i)$ witnessing (i, a_i, b_i) . We shall show that \mathbf{a} can be obtained from \mathbf{a}' , \mathbf{c} and \mathbf{c}' by applying operation φ . We need first an intermediate tuple; we define $\mathbf{d} = (d_1, \dots, d_i)$ as $\varphi(\mathbf{a}', \mathbf{c}', \dots, \mathbf{c}', \mathbf{c})$. Finally we obtain $\mathbf{e} = (e_1, \dots, e_i)$ as $\varphi(\mathbf{a}', \mathbf{a}', \dots, \mathbf{a}', \mathbf{d})$.

Let us see that the tuple obtained \mathbf{e} is indeed \mathbf{a} . For each $l \in [i - 1]$, if $\{a_l, c_l\}$ is a majority pair then $d_l = \varphi(a_l, c_l, \dots, c_l, c_l) = c_l$, and consequently $e_l = \varphi(a_l, a_l, \dots, a_l, c_l) = a_l$. Otherwise, if $\{a_l, c_l\}$ is a minority pair, then $d_l = \varphi(a_l, c_l, \dots, c_l) = a_l$ and consequently $e_l = \varphi(a_l, a_l, \dots, a_l, a_l) = a_l$. Finally, let us look at the value of e_i . Since $\{a_i, b_i\}$ is a minority pair we have that $d_i = \varphi(b_i, b_i, \dots, b_i, a_i) = a_i$ and hence $e_i = \varphi(b_i, b_i, \dots, b_i, a_i) = a_i$ and we are done. \blacksquare

4. PROOF OF THEOREM 2.7

We prove Theorem 2.7 by giving a polynomial-time algorithm that decides correctly whether a CSP($\text{Inv}(\varphi)$) instance has a solution. The structure of the algorithm mimics that of [7].

Let $\mathcal{P} = (\{v_1, \dots, v_n\}, A, \{C_1, \dots, C_m\})$ be a CSP($\text{Inv}(\varphi)$) 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

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

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 3. Once a compact representation R'_0 of R_0 has been obtained 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 means of a call to procedure `Next`, which constitutes the core of the algorithm. The algorithm then, goes as follows:

Algorithm GMM(($\{v_1, \dots, v_n\}, A, \{C_1, \dots, C_m\}$))

Step 1 **set** R'_0 as in Example 3

Step 2 **for each** $l \in \{0, \dots, m - 1\}$ **do**

 (let C_{l+1} be $(\{v_{i_1}, \dots, v_{i_{k_{l+1}}}\}, S_{l+1})$)

Step 2.1 **set** $R'_{l+1} := \text{Next}(R'_l, i_1, \dots, i_{k_{l+1}}, S_{l+1})$

end for each

Step 3 **if** $R'_m \neq \emptyset$ **return yes**

Step 4 **otherwise return no**

Observe that if we modify step 3 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 are direct consequences of the correctness and the running time of the procedure `Next`: As it is shown in Section 4.3 (Lemma 4.1) at each iteration of Step 2.1, the call `Next`($R'_l, i_1, \dots, i_{l+1}, S_{l+1}$) correctly computes a compact representation of the relation $\{\mathbf{t} \in R_l : \text{pr}_{i_1, \dots, i_{l+1}} \mathbf{t} \in S_{l+1}\}$ which is indeed R_{l+1} . Furthermore the cost of the call is polynomial in n , $|A|$, and $|S_{l+1}|$, which gives as a total running time for the algorithm polynomial (see Corollary 4.2 for a rough approximation of the running time) in the size of the input. This finishes the proof of the correctness and time complexity of the algorithm, and hence, of Theorem 2.7. \blacksquare

Let us remark that our emphasis here is on simplicity rather than in efficiency. In fact, much better time bounds than the ones provided in our analysis can be obtained by improving the code using data structures and by performing a more accurate analysis of the running time.

The remainder 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 procedures, namely `Nonempty` and `Fix-values`, which will be intensively used by our procedure `Next`.

4.1. Procedure `Nonempty`. This procedure receives as input a k -order 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 also invariant under φ . The output of the procedure is either an n -ary tuple $\mathbf{t} \in R$ such that $\text{pr}_{i_1, \dots, i_j} \mathbf{t} \in S$ or “no” meaning that such a tuple does not exist.

Procedure `Nonempty`(R', i_1, \dots, i_j, S)

Step 1 **set** $U := R'$

Step 2 **while** $\exists \mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_k \in U$ such that
 $\text{pr}_{i_1, \dots, i_j} \varphi(\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_k) \notin \text{pr}_{i_1, \dots, i_j} U$ **do**

Step 2.1 **set** $U := U \cup \{\varphi(\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_k)\}$
endwhile

Step 3 **if** $\exists \mathbf{t}$ in U such that $\text{pr}_{i_1, \dots, i_j} \mathbf{t} \in S$ **then return** \mathbf{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 $\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_k$ that previously belong to U . Therefore, since R is invariant under φ , we can conclude that during all the execution of the procedure $U \subseteq R$. Consequently, if a tuple \mathbf{t} is returned in step 3, then it belongs to R and also satisfies that $\text{pr}_{i_1, \dots, i_j} \mathbf{t} \in S$, as desired. It only remains to show that if a “no” is returned in step 4 then there does not exist any tuple \mathbf{t} in R such that $\text{pr}_{i_1, \dots, i_j} \mathbf{t} \in S$. 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

$$\langle \text{pr}_{i_1, \dots, i_j} U \rangle = \text{pr}_{i_1, \dots, i_j} \langle U \rangle = \text{pr}_{i_1, \dots, i_j} R$$

By the condition on the “while” of step 2 we have that when the procedure leaves the execution of step 2 it is necessarily the case that for all $\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_k \in U$, $\text{pr}_{i_1, \dots, i_j} \varphi(\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_k) \in \text{pr}_{i_1, \dots, i_j} U$ and consequently $\text{pr}_{i_1, \dots, i_j} U = \langle \text{pr}_{i_1, \dots, i_j} U \rangle = \text{pr}_{i_1, \dots, i_j} R$. Hence, if there exists some \mathbf{t} in R such that $\text{pr}_{i_1, \dots, i_j} \mathbf{t} \in S$ then it must exist some \mathbf{t}' in U such that $\text{pr}_{i_1, \dots, i_j} \mathbf{t}' \in S$ 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 iteration is basically dominated by the amount of computational time needed to check whether there exists some tuples $\exists \mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_k \in U$ such that $\text{pr}_{i_1, \dots, i_j} \varphi(\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_k) \notin \text{pr}_{i_1, \dots, i_j} U$ in step 2. In order to try all possible combinations for $\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_k$ in U , $|U|^k$ steps suffice. Each one of these steps requires time $O(|U|n)$, as tuples have arity n and checking whether $\varphi(\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_k)$ belongs to U can be done

naively by a sequential search in U . Thus, the total running time of step 2 is $O(|U|^{k+1}n)$. The cost of step 3 is the cost of finding a tuple \mathbf{t} in U satisfying $\text{pr}_{i_1, \dots, i_j} \mathbf{t} \in S$ which is $O(|U||S|n)$. Putting all together we obtain that the complete running time of the procedure is $O(|U|^{k+2}n + |U||S|n)$ which we can bound by $O(|U|^{k+2}|S|n)$. So, it only remains to bound the size of U (at the end of the execution of the procedure). The size of U can be bounded by the initial size of R' which is at most $O(n|A|^2 + n^k|A|^k) = O((n|A|)^k)$ (since R' is compact) plus the number of iterations in step 2, which is bounded by $|\text{pr}_{i_1, \dots, i_j} R|$.

Consequently the total running time of the procedure can be bounded by

$$O\left(\left((n|A|)^k + |\text{pr}_{i_1, \dots, i_j} R|\right)^{k+2} |S|n\right).$$

We want to remark here that the size of $\text{pr}_{i_1, \dots, i_j} R$ can be exponentially large in the size of the input. For now we do not deal with this issue. Later we shall see how, in order to overcome this difficulty, we organize invoking to `Nonempty` in such a way that the value of $\text{pr}_{i_1, \dots, i_j} R$ is conveniently bounded.

4.2. Procedure Fix-values. This procedure receives as 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

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

Figure 1 contains a description of the procedure. Let us study its correctness. We shall show by induction on $j \in \{0, \dots, m\}$ that U_j is a compact representation of $R_j = \{\mathbf{t} \in R : \text{pr}_1 \mathbf{t} = a_1, \dots, \text{pr}_j \mathbf{t} = a_j\}$. 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} . We shall start by showing that at the end of the execution of step 2.2, $\text{Sig}_{U_{j+1}} = \text{Sig}_{R_{j+1}}$. 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 $\text{Sig}_{R_{j+1}}$. So it only remains to see that when the “if” in step 2.2.1 is satisfied, we have that (a) $(\mathbf{t}_1, \mathbf{t}_5)$ witnesses (i, a, b) , and (b) \mathbf{t}_1 and \mathbf{t}_5 are tuples in R_{j+1} ,

Proof of (a): We shall first show that for each $l \in [i-1]$, $\text{pr}_l \mathbf{t}_1 = \text{pr}_l \mathbf{t}_5$. Let c_l to be $\text{pr}_l \mathbf{t}_1$ and let $d_l = \text{pr}_l \mathbf{t}_2 = \text{pr}_l \mathbf{t}_3$. If $\{c_l, d_l\}$ is a majority pair then $\text{pr}_l \mathbf{t}_4 = \varphi(c_l, d_l, \dots, d_l, d_l) = d_l$ and hence $\text{pr}_l \mathbf{t}_5 = \varphi(c_l, c_l, \dots, c_l, d_l) = c_l$. Otherwise, if $\{c_l, d_l\}$ is a minority pair, then $\text{pr}_l \mathbf{t}_4 = \varphi(c_l, d_l, \dots, d_l, d_l) = c_l$ and consequently $\text{pr}_l \mathbf{t}_5 = \varphi(c_l, c_l, \dots, c_l, c_l) = c_l$. So it only remains to show that $\text{pr}_i \mathbf{t}_1 = a$ and $\text{pr}_i \mathbf{t}_5 = b$. We have $\text{pr}_i \mathbf{t}_1 = a$ as a direct consequence of the fact that \mathbf{t}_1 is the tuple returned by the call to `Nonempty` $(U_j, j + 1, i, \{(a_{j+1}, a)\})$. Observe also that as $(\mathbf{t}_2, \mathbf{t}_3)$ witnesses (i, a, b) we have that $\text{pr}_i \mathbf{t}_2 = a$ and $\text{pr}_i \mathbf{t}_3 = b$. Consequently, since $\{a, b\}$ is a minority pair we have that $\text{pr}_i \mathbf{t}_4 = \varphi(a, a, \dots, a, b) = b$ and hence $\text{pr}_i \mathbf{t}_5 = \varphi(a, a, \dots, a, b) = b$, as desired.

Proof of (b): As \mathbf{t}_1 is the output of the call `Nonempty` $(U_j, j + 1, i, \{(a_{j+1}, a)\})$, we can conclude that \mathbf{t}_1 belongs to R_j , $\text{pr}_{j+1} \mathbf{t}_1 = a_{j+1}$, and $\text{pr}_i \mathbf{t}_1 = a$. 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 , we can conclude that \mathbf{t}_5 belongs to R_j . Thus in order to see that \mathbf{t}_5 belongs to R_{j+1} it only remains to show that $\text{pr}_{j+1} \mathbf{t}_5 = a_{j+1}$. This can be obtained as a direct consequence of (a), since as $(\mathbf{t}_1, \mathbf{t}_5)$ witnesses (i, a, b) and $i > j + 1$, we have that $a_{j+1} = \text{pr}_{j+1} \mathbf{t}_1 = \text{pr}_{j+1} \mathbf{t}_5$.

We have just seen that at the end of step 2.2, $\text{Sig}_{U_{j+1}} = \text{Sig}_{R_{j+1}}$. In Step 2.3, procedure `Fix-values` enlarges U_{j+1} so that for every set $I \subseteq [n]$ with $|I| \leq k - 1$, $\text{pr}_I U_{j+1} =$

```

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  $(i, a, b) \in [n] \times A^2$ ,  $\{a, b\}$  is a minority pair, do
Step 2.2.1  if Nonempty( $U_j, j + 1, i, \{(a_{j+1}, a)\}$ )  $\neq$  "no" and
               $(i, a, b) \in \text{Sig } U_j$  and  $i > j + 1$  then
                (let  $\mathbf{t}_1$  be the tuple returned by Nonempty( $U_j, j + 1, i, \{(a_{j+1}, a)\}$ )
                  and let  $\mathbf{t}_2, \mathbf{t}_3$  be tuples in  $U_j$  witnessing  $(i, a, b)$  )
Step 2.2.1.1 set  $\mathbf{t}_4 := \varphi(\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_2, \mathbf{t}_3)$ 
Step 2.2.1.2 set  $\mathbf{t}_5 := \varphi(\mathbf{t}_1, \mathbf{t}_1, \dots, \mathbf{t}_1, \mathbf{t}_4)$ 
Step 2.2.1.3 set  $U_{j+1} := U_{j+1} \cup \{\mathbf{t}_1, \mathbf{t}_5\}$ 
              end for each
Step 2.3    for each  $k' \in [k - 1]$ 
              for each  $l_1, \dots, l_{k'} \in [n]$  with  $l_1 < l_2 < \dots < l_{k'}$ 
              for each  $d_1, \dots, d_{k'} \in A$  do
Step 2.3.1  if Nonempty( $U_j, l_1, \dots, l_{k'}, j + 1, \{(d_1, \dots, d_{k'}, a_{j+1})\}$ )  $\neq$  "no" then
                (let  $\mathbf{t}_6$  be the tuple returned by the call
                  to Nonempty( $U_j, l_1, \dots, l_{k'}, j + 1, \{(d_1, \dots, d_{k'}, a_{j+1})\}$ ))
                set  $U_{j+1} := U_{j+1} \cup \{\mathbf{t}_6\}$ 
              end for each
Step 2.4    set  $j := j + 1$ 
            end while
Step 3      return  $U_m$ 

```

Figure 1: Fix-values

$\text{pr}_I R_{j+1}$. The proof of this fact is rather straightforward. Let $k', l_1, \dots, l_{k'}, d_1, \dots, d_{k'}$ be the running parameters of a given iteration of step 2.3. It is easy to observe that the call to Nonempty($U_j, l_1, \dots, l_{k'}, j + 1, \{(d_1, \dots, d_{k'}, a_{j+1})\}$) is different than "no" if and only if $(d_1, \dots, d_{k'}) \in \text{pr}_{l_1, \dots, l_{k'}} R_{j+1}$. Furthermore, if the call returns a tuple \mathbf{t}_6 then we can guarantee that \mathbf{t}_6 belongs to R_{j+1} and that $\text{pr}_I \mathbf{t}_6 = (d_1, \dots, d_{k'})$. We have just seen that at the end of the execution of step 2.3 for every set $I \subseteq [n]$ with $|I| \leq k - 1$, $\text{pr}_I U_{j+1} = \text{pr}_I R_{j+1}$.

Notice that at each iteration of step 2.2, at most 2 tuples are added for each (i, a, b) in $\text{Sig}_{R_{j+1}}$. Furthermore at each iteration of step 2.3, at most one tuple is added per each k', I with $|I| \leq k - 1$ and tuple in $\text{pr}_I R_{j+1}$. Consequently, U_{j+1} is compact. This completes the proof of its correctness.

Let us study now its time complexity. The "while" loop at step 2 is performed $m \leq n$ times. At each iteration the procedure executes two loops (Step 2.2 and Step 2.3). The "for each" loop at step 2.2 is executed for each (i, a, b) in $[n] \times A^2$ with $\{a, b\}$ a minority pair. That is a total number of times bounded by $n|A|^2$. The cost of each iteration of the loop in Step 2.2 is basically dominated by the cost of the call to procedure Nonempty which costs $O(((n|A|)^k + |A|^2)^{k+2}n)$ which is $O((n|A|)^{k(k+2)+1})$. The "for each" loop at step 2.3 is executed $O((n|A|)^k)$ times. The cost of each iteration is basically the cost of the call to Nonempty which is $O(((n|A|)^k + |A|^k)^{k+2}n) = O((n|A|)^{k(k+2)+1})$. Thus the total cost of

```

Procedure Next-beta( $R', i_1, \dots, i_j, S$ )
Step 1  set  $U := \emptyset$ 
Step 2  for each  $(i, a, b) \in [n] \times A^2$ ,  $\{a, b\}$  is a minority pair do
Step 2.1 if Nonempty( $R', i_1, \dots, i_j, i, S \times \{a\}$ )  $\neq$  "no" then
        (let  $\mathbf{t}_1$  be Nonempty( $R', i_1, \dots, i_j, i, S \times \{a\}$ ))
Step 2.2 if Nonempty(Fix-values( $R', \text{pr}_1 \mathbf{t}_1, \dots, \text{pr}_{i-1} \mathbf{t}_1$ ),  $i_1, \dots, i_j, i, S \times \{b\}$ )  $\neq$  "no"
        (let  $\mathbf{t}_2$  be Nonempty(Fix-values( $R', \text{pr}_1 \mathbf{t}, \dots, \text{pr}_{i-1} \mathbf{t}$ ),  $i_1, \dots, i_j, i, S \times \{b\}$ ))
        set  $U := U \cup \{\mathbf{t}_1, \mathbf{t}_2\}$ 
    end for each
Step 3  for each  $k' \in [k-1]$ 
        for each  $l_1, \dots, l_{k'} \in [n]$  with  $l_1 < l_2 < \dots < l_{k'}$ 
        for each  $d_1, \dots, d_{k'} \in A$  do
Step 3.1 if Nonempty( $R', i_1, \dots, i_j, l_1, \dots, l_{k'}, S \times \{(d_1, \dots, d_{k'})\}$ )  $\neq$  "no" then
        (let  $\mathbf{t}_3 =$  Nonempty( $R', i_1, \dots, i_j, l_1, \dots, l_{k'}, S \times \{(d_1, \dots, d_{k'})\}$ ))
        set  $U := U \cup \{\mathbf{t}_3\}$ 
Step 4  return  $U$ 

```

Figure 2: Next-beta

step 2.3 is $O((n|A|)^{k(k+2)+1+k})$. Thus the combined cost of steps 2.2 and 2.3 is dominated by the cost of 2.3 which gives as a total cost of $O((n|A|)^{k(k+2)+1+k})$ for each iteration of step 2. Since step 2 is executed at most n times we have a total cost of the procedure of $O((n|A|)^{k(k+2)+1+k}n)$ which we shall bound by $O((n|A|)^{(k+1)(k+2)})$.

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 $R^* = \{\mathbf{t} \in R : \text{pr}_{i_1, \dots, i_j} \mathbf{t} \in S\}$. It is an easy exercise to verify that R^* must also be invariant under φ .

We shall start by defining a procedure, called **Next-beta** that although equivalent to **Next** has a worse running time. In particular, the running time of **Next-beta** might be exponential to the arity j of S (recall that arities can be unbounded as we allow infinite constraint languages).

Figure 2 contains a description of the procedure. The overall structure of procedure **Next-beta** is similar to that of procedure **Fix-values**. The procedure constructs a representation U of R^* . Initially U is empty. In step 2, **Next-beta** adds tuples to U so that when it leaves the execution of step 2, $\text{Sig}_U = \text{Sig}_{R^*}$. Let us analyze step 2. Observe that the condition of the “if” statement

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

of step 2.1 is satisfied if and only if there exists a tuple $\mathbf{t}_1 \in R$ such that $\text{pr}_{i_1, \dots, i_j} \mathbf{t}_1 \in S$ and $\text{pr}_i \mathbf{t}_1 = a$. 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

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

This condition is satisfied if and only if there exists some \mathbf{t}_2 in R such that $\text{pr}_{i_1, \dots, i_j} \mathbf{t}_2 \in S$, $\text{pr}_{1, \dots, i-1} \mathbf{t}_2 = \text{pr}_{1, \dots, i-1} \mathbf{t}_1$ and $\text{pr}_i \mathbf{t}_2 = b$. It is immediate to see that if the condition holds then $\mathbf{t}_2 \in R^*$ and $(\mathbf{t}_1, \mathbf{t}_2)$ witnesses (i, a, b) . It only remains to show that if $(i, a, b) \in \text{Sig}_{R^*}$ then such a \mathbf{t}_2 must exist: Let $\mathbf{t}_a, \mathbf{t}_b$ be tuples in R^* witnessing (i, a, b) and let \mathbf{t}_1 be the tuple returned by the call to procedure **Nonempty** in step 2.1. In order to prove the existence of \mathbf{t}_2 we shall use the usual trick. First define a tuple \mathbf{u} as $\varphi(\mathbf{t}_1, \mathbf{t}_a, \dots, \mathbf{t}_a, \mathbf{t}_b)$ and finally let us define \mathbf{t}_2 as $\varphi(\mathbf{t}_1, \mathbf{t}_1, \dots, \mathbf{t}_1, \mathbf{u})$. Since $\mathbf{t}_1, \mathbf{t}_a, \mathbf{t}_b$ belong to R^* and R^* is invariant under φ we can conclude that \mathbf{t}_2 belongs to R^* . Let us show that $\text{pr}_{1, \dots, i-1} \mathbf{t}_2 = \text{pr}_{1, \dots, i-1} \mathbf{t}_1$: For each $l \in [i-1]$, let c_l be $\text{pr}_l \mathbf{t}_1$ and let d_l be $\text{pr}_l \mathbf{t}_a = \text{pr}_l \mathbf{t}_b$. If $\{c_l, d_l\}$ is a majority pair then $\text{pr}_l \mathbf{u} = \varphi(c_l, d_l, \dots, d_l, d_l) = d_l$ and hence $\text{pr}_l \mathbf{t}_2 = \varphi(c_l, c_l, \dots, c_l, d_l) = c_l$. Otherwise, $\{c_l, d_l\}$ is a minority pair and hence $\text{pr}_l \mathbf{u} = \varphi(c_l, d_l, \dots, d_l, d_l) = c_l$. Consequently, $\text{pr}_l \mathbf{t}_2 = \varphi(c_l, c_l, \dots, c_l, c_l) = c_l$ and we are done. Finally we need to see that $\text{pr}_i \mathbf{t}_2 = b$. Observe that since $\{a, b\}$ is a minority pair we have that $\text{pr}_i \mathbf{u} = \varphi(a, a, \dots, a, b) = b$ and hence $\text{pr}_i \mathbf{t}_2 = \varphi(a, a, \dots, a, b) = b$.

We have just proved that, if U is the representation output by the procedure in Step 4, then $\text{Sig}_U = \text{Sig}_{R^*}$. It is straightforward to verify that step 3 guarantees that for each I with $|I| \leq k-1$, $\text{pr}_I U = \text{pr}_I R^*$. The analysis here is basically identical to that of step 2.3 in procedure **Fix-values**. Consequently, U is a representation of R^* .

At each iteration of step 2, at most 2 tuples are added for each (i, a, b) in Sig_{R^*} . Furthermore at each iteration of step 3, at most one tuple is added per each k' , I , with $|I| \leq k-1$ and tuple in $\text{pr}_I R_{j+1}$. Consequently, U is compact. This completes the proof of its correctness.

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. The cost of calling $\text{Nonempty}(R', i_1, \dots, i_j, i, S \times \{a\})$ in step 2.1 is $O(((n|A|)^k + r)^{k+2} |S|n)$ where r is $|\text{pr}_{i_1, \dots, i_j, i} R|$.

The cost of calling

$$\text{Nonempty}(\text{Fix-values}(R', \text{pr}_1 \mathbf{t}_1, \dots, \text{pr}_{i-1} \mathbf{t}_1), \\ i_1, \dots, i_j, i, S \times \{b\})$$

in step 2.2 is the sum of the call to **Fix-values** which is $O((n|A|)^{(k+1)(k+2)})$ and the call to **Nonempty** which is $O(((n|A|)^k + r)^{k+2} |S|n)$. Therefore, the total cost of an iteration of the loop of step 2 is

$$O\left(\left((n|A|)^k + r\right)^{k+2} |S|n + (n|A|)^{(k+1)(k+2)}\right)$$

Hence, in order to obtain the total running time for the procedure we only need to multiply the previous quantify by the number of iterations, which is $n|A|^2$, obtaining

$$n|A|^2 \left(\left((n|A|)^k + r \right)^{k+2} |S|n + (n|A|)^{(k+1)(k+2)} \right)$$

Let us take a closer look at the value of $r = |\text{pr}_{i_1, \dots, i_j, i} R|$. It is important to notice here that the set of possible constraints S that can appear in an instance is *infinite* and henceforth it is not possible to bound the value of j . Consequently, the value of r might

be exponential in the worst case. However, it would be possible to bound the value of j and get a polynomial bound for r if a *finite* subset Γ of $\text{Inv}(\varphi)$ is fixed beforehand and we assume that all constraint instances use only constraint relations from Γ . Such a situation is not completely unusual. In fact, a good number of results on the complexity of $\text{CSP}(\Gamma)$ including the pioneering work of Schaeffer [26] assumes Γ to be finite. By using the procedure **Next-beta** it could be possible to define a polynomial-time algorithm that solves $\text{CSP}(\Gamma)$ for every finite subset Γ of $\text{Inv}(\varphi)$. However we are aiming here for a more general result. To this end, we define a new procedure **Next** which makes a sequence of calls to **Next-beta**.

Procedure $\text{Next}(R', i_1, \dots, i_j, S)$

Step 1 **set** $l := 0, U_l := R'$

Step 2 **while** $l < j$ **do**

Step 2.1 **set** $U_{l+1} := \text{Next-beta}(U_l, i_1, \dots, i_{l+1}, \text{pr}_{1, \dots, l+1} S)$

Step 2.2 **set** $l := l + 1$

end while

Step 3 **return** U_j

Observe that at each call of the procedure **Next-beta** in step 2.1, the value of r can be bounded by $|\text{pr}_{1, \dots, l} S||A|$, and hence the running time of each call to **Next-beta** can be bounded (very grossly) by

$$O\left(n^{(k+1)(k+2)+1}|A|^{(k+1)(k+2)+2}|S|^{k+3}\right)$$

Finally the running time of **Next** is obtained by multiplying by n (which always bounds j) the previous quantity.

Lemma 4.1. *For every $n \geq 1$, every n -ary relation R invariant under φ , every compact representation R' of R , every $i_1, \dots, i_j \in [n]$, and every j -ary relation S invariant under φ , $\text{Next}(R', i_1, \dots, i_j, S)$ computes a compact representation of $R^* = \{\mathbf{t} \in R : \text{pr}_{i_1, \dots, i_j} \mathbf{t} \in S\}$ in time $O((n|A|)^{(k+1)(k+2)+2}|S|^{k+3})$. Furthermore R^* is invariant under φ . ■*

Finally, we have

Corollary 4.2. *Algorithm **Solve** decides correctly if an instance \mathcal{P} of $\text{CSP}(\text{Inv}(\varphi))$ is satisfiable in time $O(m(n|A|)^{(k+1)(k+2)+2}|S^*|^{k+3})$ where n is the number of variables of \mathcal{P} , m is its number of constraints and S^* is the largest constraint relation occurring in \mathcal{P} . ■*

5. ACKNOWLEDGMENTS

Research partially supported by the MCyT under grants TIC 2002-04470-C03, TIC 2002-04019-C03, TIN 2004-04343, the EU PASCAL Network of Excellence, IST-2002-506778, and the MODNET Marie Curie Research Training Network, MRTN-CT-2004-512234.

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 Symposium on Logic in Computer Science, (LICS'03)*, pages 321–330, 2003.

- [4] A. Bulatov. A Graph of a Relational Structure and Constraint Satisfaction Problems. In *Proceedings of the 19th IEEE Annual Symposium on Logic in Computer Science, (LICS'04)*, pages 448–457, 2004.
- [5] A. Bulatov. Combinatorial problems raised from 2-semilattices. *Journal of Algebra*, to appear.
- [6] 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.
- [7] A. Bulatov and V. Dalmau. A Simple Algorithm for Mal'tev Constraints To appear in SIAM J. Computing.
- [8] A. Bulatov, P. Jeavons, and M. Volkov. Finite Semigroups Imposing Tractable Constraints. In *School on Algorithmic Aspects of the Theory of Semigroups*, 2001.
- [9] A. Bulatov, A. Krokhin, and P. Jeavons. Constraint Satisfaction Problems and Finite Algebras. In *Proceedings of the 27th International Colloquium on Automata Languages, and Programming, (ICALP'00)*, volume 1853 of *Lecture Notes in Computer Science*, pages 160–171, 2000.
- [10] A. Bulatov, A. Krokhin, and P. Jeavons. The Complexity of Maximal Constraint Languages. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, (STOC'01)*, pages 667–674, 2001.
- [11] H.M. Chen and V. Dalmau. (Smart) Look-Ahead Arc Consistency and the Pursuit of CSP Tractability. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming, (CP'04)*, 2004.
- [12] S.A. Cook. The Complexity of Theorem-Proving Procedures. In *3rd Annual ACM Symposium on Theory of Computing, (STOC'71)*, pages 151–158, 1971.
- [13] M.C. Cooper, D.A. Cohen, and P.G. Jeavons. Characterizing Tractable Constraints. *Artificial Intelligence*, 65:347–361, 1994.
- [14] V. Dalmau. A New Tractable Class of Constraint Satisfaction Problems. *Ann. Math. Artif. Intell.*, 44(1-2):61–85, 2005
- [15] V. Dalmau, P. Kolaitis, and M. Vardi. Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics In *8th International Conference on Principles and Practice of Constraint Programming, (CP'02)*, volume 2470 of *Lecture Notes in Computer Science*, pages 310–326, Berlin/New York, 2002. Springer-Verlag.
- [16] V. Dalmau, R. Gavaldà, P. Tesson, and D. Thérien. Tractable Clones of Polynomials over Semigroups In *11th International Conference on Principles and Practice of Constraint Programming, (CP'05)*, volume 3709 of *Lecture Notes in Computer Science*, pages 196–210, Berlin/New York, 2005. Springer-Verlag.
- [17] 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.
- [18] T. Feder. Constraint satisfaction on finite groups with near subgroups. Manuscript.
- [19] 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. Computing*, 28(1):57–104, 1998.
- [20] M. Grohe. The Complexity of Homomorphism and Constraint Satisfaction Problems seen from the Other Side. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science, (FOCS'03)*, pages 552–561, 2003.
- [21] P. Jeavons. On the Algebraic Structure of Combinatorial Problems. *Theoretical Computer Science*, 200:185–204, 1998.
- [22] P. Jeavons, D. Cohen, and M.C. Cooper. Constraints, Consistency and Closure. *Artificial Intelligence*, 101:251–265, 1998.
- [23] P. Jeavons, D. Cohen, and M. Gyssens. Closure Properties of Constraints. *Journal of the ACM*, 44(4):527–548, July 1997.
- [24] L. Kirousis. Fast Parallel Constraint Satisfaction. *Artificial Intelligence*, 64:147–160, 1993.
- [25] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
- [26] 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.