Conjunctive-Query Containment and Constraint Satisfaction*

Phokion G. Kolaitis†
Computer Science Department
University of California, Santa Cruz
Santa Cruz, CA 95064, U.S.A.
kolaitis@cse.ucsc.edu

Moshe Y. Vardi‡
Department of Computer Science
Rice University
Houston, TX 77005-1892, U.S.A.
vardi@cs.rice.edu

Abstract

Conjunctive-query containment is recognized as a fundamental problem in database query evaluation and optimization. At the same time, *constraint satisfaction* is recognized as a fundamental problem in artificial intelligence. What do conjunctive-query containment and constraint satisfaction have in common? Our main conceptual contribution in this paper is to point out that, despite their very different formulation, conjunctive-query containment and constraint satisfaction are essentially the same problem. The reason is that they can be recast as the following fundamental algebraic problem: given two finite relational structures A and B, is there a homomorphism $h : A \to B$? As formulated above, the homomorphism problem is *uniform* in the sense that both relational structures A and B are part of the input. By fixing the structure B, one obtains the following *non-uniform* problem: given a finite relational structure A, is there a homomorphism $h : A \to B$? In general, non-uniform tractability results do not uniformize. Thus, it is natural to ask: which tractable cases of non-uniform tractability results for constraint satisfaction and conjunctive-query containment do uniformize?

Our main technical contribution in this paper is to show that several cases of tractable non-uniform constraint satisfaction problems do indeed uniformize. We exhibit three non-uniform tractability results that uniformize and, thus, give rise to polynomial-time solvable cases of constraint satisfaction and conjunctive-query containment. We begin by examining the tractable cases of Boolean constraint-satisfaction problems and show that they do uniformize. This can be applied to conjunctive-query containment via *Booleanization*; in particular, it yields one of the known tractable cases of conjunctive query containment. After this, we show that tractability results for constraint-satisfaction problems that can be expressed using Datalog programs with

---

*A preliminary version of this paper appeared in *Proc. 17th ACM Symp. on Principles of Database Systems*, June 1998, pp. 205–213.

†Partially supported by NSF grant CCR-9610257.

‡Partially supported by NSF grants CCR-9628400 and CCR-9700061. URL: http://www.cs.rice.edu/~vardi/

bounded number of distinct variables also uniformize. Finally, we provide a new proof for the fact that tractability results for queries with bounded *treewidth* uniformize as well, via a connection with first-order logic with a bounded number of distinct variables.

1 Introduction

Conjunctive queries have had a conspicuous presence in both the theory and practice of database systems since the 1970s. Conjunctive queries constitute a broad class of frequently used queries, because their expressive power is equivalent to that of the Select-Join-Project queries in relational algebra [AHV95, Ull89]. For this reason, several algorithmic problems concerning conjunctive queries have been investigated in depth. In particular, *conjunctive-query containment* was recognized fairly early as a fundamental problem in database query evaluation and optimization. Indeed, conjunctive-query containment is essentially the same problem as conjunctive query evaluation; moreover, conjunctive-query containment can be used as a tool in query optimization, since query equivalence is reducible to query containment.

Chandra and Merlin [CM77] studied the computational complexity of conjunctive query containment and showed that it is an NP-complete problem. In recent years, there has been renewed interest in the study of conjunctive query containment, because of its close relationship to the problem of answering queries using materialized views [LMSS95, RSU95]. The latter has emerged as a central problem in integrating information from heterogeneous sources, an area that lately has been the focus of concentrated research efforts (see [Ull97] for survey). Since conjunctive-query containment is intractable in its full generality, researchers have embarked on a search for tractable cases. These are obtained by imposing syntactic or structural restrictions on the conjunctive queries Q_1 and Q_2 that serve as input to the problem: "is $Q_1 \subseteq Q_2$?". In particular, Saraiya [Sar91] showed that conjunctive-query containment can be solved in linear time, if every database predicate occurs at most twice in the body of Q_1. More recently, Chekuri and Rajaraman [CR97, CR98] showed that, for every $k \geq 1$, conjunctive-query containment can be solved in polynomial time, if Q_2 has *querywidth* at most k and a *query decomposition* of Q_2 of width k is available. The concept of querywidth is closely related to the well-studied concept of *treewidth* of a graph (see [vL90, Bod93]). It should be noted that queries of width 1 are precisely the *acyclic* queries; thus, Chekuri and Rajaraman's results extend the earlier work of Yannakakis [Yan81] and Qian [Qia96] on query evaluation and containment for acyclic queries.

Starting with the pioneering work of Montanari [Mon74] researchers in artificial intelligence have investigated a class of combinatorial problems that became known as *constraint-satisfaction problems* (CSP). The input to such a problem consists of a set of variables, a set of possible values for the variables, and a set of constraints between the variables; the question is to determine whether there is an assignment of values to the variables that satisfies the given constraints. The study of constraint satisfaction occupies a prominent place in artificial intelligence, because many problems that arise in different areas can be modeled as constraint-satisfaction problems in a natural way; these areas include Boolean satisfiability, temporal reasoning, belief maintenance, machine vision, and scheduling (see [Dec92, Kum92, Mes89, Tsa93]). In its full generality, constraint satisfaction is an NP-complete problem. For this reason, researchers in artificial intelligence have pursued both heuristics for constraint-satisfaction problems and tractable cases obtained by imposing restrictions on the constraints (see [MF93, Dec92, PJ97]).

What do conjunctive-query contaiment and constraint satisfaction have in common? Despite their very different formulation, it turns out that conjunctive-query containment and constraint satisfaction are essentially the same problem. The reason is that they can be recast as the following fundamental algebraic problem: given two finite relational structures A and B, is there a homomorphism $h : A \to B$? Indeed, on the side of conjunctive-query containment, it is well known that $Q_1 \subseteq Q_2$ if and only if there is a homomorphism $h : Q_2^D \to Q_1^D$, where Q_i^D is the *canonical* database associated with the query Q_i, $i = 1, 2$ [CM77]. On the side of constraint satisfaction, a perusal of the literature reveals that all constraint-satisfaction problems studied can be viewed as special cases of the above homomorphism problem [FV93, FV99] (see also [Jea97]). It should be noted that several researchers, including [Bib88, Dec90, GJC94, PJ97], have observed that there are tight connections between constraint-satisfaction problems and certain problems in relational databases. In particular, Gyssens, Jeavons and Cohen [GJC94] pointed out that the set of all solutions to a constraint satisfaction problem coincides with the join of certain relations extracted from the given constraint-satisfaction problem. Thus, solving constraint-satisfaction problems and evaluating joins are interreducible. In turn, conjunctive-query evaluation and join valuation are also reducible to each other [Ull89]. Since Chandra and Merlin [CM77] showed that conjunctive query evaluation and conjunctive query containment are equivalent problems, this provides a different (although less direct) way to establish the tight connection between conjunctive query containment and constraint satisfaction. Nonetheless, it is fair to say that overall there is little interaction between the community that pursues tractable cases of conjunctive query containment and the community that pursues tractable cases of constraint satisfaction. One of our main goals in this paper is to make the connection between conjunctive query containment and constraint satisfaction explicit, bring it to front stage and, thus, further enhance the interaction between database theory and artificial intelligence.

As formulated above, the homomorphism problem is *uniform* in the sense that both relational structures A and B are part of the input. By fixing the structure B, one obtains the following *non-uniform* problem CSP(B): given a finite relational structure A, is there a homomorphism $h : A \to B$? Over the past twenty years, researchers in computational complexity have studied such non-uniform problems in an attempt to determine for which structures B the associated CSP(B) problem is tractable and for which it is intractable. The first remarkable success on this front was obtained by Schaefer [Sch78], who pinpointed the computational complexity of *Boolean* CSP(B) problems, in which the structure B is Boolean (i.e., has the set $\{0, 1\}$ as its universe). Schaefer established a *dichotomy* theorem for Boolean CSP(B) problems. Specifically, he identified six classes of Boolean structures and showed that CSP(B) is solvable in polynomial time, if B is in one of these classes, but CSP(B) is NP-complete in all other cases. Note that each Boolean CSP(B) problem can be viewed as a *generalized satisfiability problem*. In particular, Schaefer's [Sch78] dichotomy theorem provides a coherent explanation for the computational complexity of Horn Satisfiability, 2-Satisfiability, One-in-Three Satisfiability, and other such Boolean satisfiability problems. After this, Hell and Nešetřil [HN90] established a dichotomy theorem for CSP(B)

problems in which B is an *undirected* graph: if B is 2-colorable, then CSP(B) is solvable in polynomial time; otherwise, CSP(B) is NP-complete. Observe that if K_k is a clique with k nodes, then CSP(K_k) is the k-COLORABILITY problem, $k \geq 2$. Thus, Hell and Nešetřil's dichotomy theorem generalizes the results concerning the computational complexity of the k-Colorability problem for each $k \geq 2$. Motivated by these dichotomy results, Feder and Vardi [FV99] raised the question: is every CSP(B) problem either solvable in polynomial time or NP-complete? Although they did not settle this question, Feder and Vardi [FV99] were able to isolate two conditions that imply polynomial-time solvability of CSP(B) problems; moreover, they argued that all known polynomially solvable CSP(B) problems satisfy one of these conditions. The first condition asserts that the complement of the CSP(B) problem at hand is expressible in Datalog (CSP(B) itself cannot be expressible in Datalog, because it is not a monotone problem); this condition covers such known tractable cases as Horn Satisfiability, 2-Satisfiability, and 2-Colorability. The second condition is group-theoretic and covers Schaefer's [Sch78] tractable class of *affine* satisfiability problems.

In general, non-uniform tractability results do not uniformize. Thus, tractability results for each problem in a collection of non-uniform CSP(B) problems do not necessarily yield a tractable case of the uniform constraint satisfaction problem (or of the conjunctive query containment problem). The reason is that both structures A and B are part of the input to the constraint satisfaction problem, and the running times of the polynomial-time algorithms for CSP(B) may very well be exponential in the size of B. Thus, it is natural to ask: which tractable cases of non-uniform CSP(B) problems uniformize and give rise to uniform tractable cases of constraint satisfaction and, equivalently, to conjunctive-query containment.

Our main technical contribution in this paper is to show that several cases of tractable non-uniform CSP(B) problems do indeed uniformize. We begin by examining the main tractable cases of Boolean CSP(B) problems considered by Schaefer [Sch78]. These are the cases where CSP(B) corresponds to a 2-satisfiability problem, a Horn satisfiability problem, a dual Horn satisfiability problem, or an affine satisfiability problem. For all these cases, uniform polynomial-time algorithms can be obtained by combining polynomial-time algorithms that detect membership in these cases, build a corresponding Boolean formula, and apply the polynomial-time algorithm for satisfiability of such formulas. It should be pointed out, however, that the formula-building algorithms for 2-Satisfiability, Horn satisfiability, and dual Horn satisfiability are in the worst case quadratic in the size of B. In turn, this yields cubic-time algorithms for the corresponding uniform constraint-satisfaction problems. We show here that a better bound can be achieved by designing algorithms that skip the formula-building phase. Although these results are about Boolean constraint-satisfaction problems, they turn out to have applications to conjunctive-query containment. For this, we show that conjunctive-query containment problems can be *binarized* and reduced to Boolean constraint-satisfaction problems. As a concrete application, we show that Sarayia's [Sar91] tractable case of conjunctive-query containment can be derived using this technique.

After this, we focus on the connections between Datalog and constraint satisfaction. As mentioned earlier, Feder and Vardi [FV93, FV99] realized that the tractability of many non-uniform CSP(B) problems can be globally explained by the fact that the complement of each of these problems is expressible in Datalog. Using pebble-game techniques introduced in [KV95], we show here that such non-uniform tractability results uniformize, as long as

Datalog programs with a bounded number of distinct variables are considered. Specifically, we establish that, for every $k \geq 1$, there is a polynomial-time algorithm for testing whether there is a homomorphism $h : A \to B$, where A and B are two given relational structures such that the complement of $\mathrm{CSP}(B)$ is expressible by a Datalog program with at most k distinct variables in each rule.

Up to this point, we have obtained tractable cases of the constraint satisfaction problem "is there a homomorphism $h : A \to B$?" by imposing restrictions on the structure B. Our last result concerns a known tractable case of constraint satisfaction obtained by imposing restrictions on the structure A, namely the case where A is a structure of *bounded treewidth* (see Section 5 for a discussion of earlier work). Specifically, we provide a new proof for the fact that, for every $k \geq 1$, there is a polynomial-time algorithm for testing whether, given a structure A of treewidth at most k and an arbitrary structure B, there is a homomorphism from A to B. To this effect, we establish a correspondence between structures of treewidth at most k and conjunctive queries expressible in FO^{k+1}, the fragment of first-order logic with at most $k+1$ distinct variables, and then apply a polynomial-time algorithm for evaluating FO^{k+1} queries.

2 Preliminaries

Formally, a n-ary *conjunctive query* Q is a query definable by a positive existential first-order formula $\varphi(X_1, \ldots, X_n)$ having conjunction as its only Boolean connective, that is, by a formula of the form

$$(\exists Z_1) \ldots (\exists Z_m) \psi(X_1, \ldots, X_n, Z_1, \ldots, Z_m),$$

where $\psi(X_1, \ldots, X_n, Z_1, \ldots, Z_m)$ is a conjunction of extensional database predicates. The free variables $X_1, \ldots, X_n$ of the definining formula are called the *distinguished variables* of Q. Such a conjunctive query is usually written as rule, whose head is $Q(X_1, \ldots, X_n)$ and whose body is $\psi(X_1, \ldots, X_n, Z_1, \ldots, Z_m)$. For example, the formula

$$(\exists Z_1 \exists Z_2)(P(X_1, Z_1, Z_2) \wedge R(Z_2, Z_3) \wedge R(Z_3, X_2)$$

defines a conjunctive query Q, which as a rule becomes

$$Q(X_1, X_2) \text{ :- } P(X_1, Z_1, Z_2), R(Z_2, Z_3), R(Z_3, X_2).$$

If D is a database, then $Q(D)$ is the n-ary relation on D obtained by evaluating the query Q on D, that is, the collection of all n-tuples from D that satisfy the query. Note that we need to choose an order for the free variables. In the example above we chose the order X_1, X_2, but the order X_2, X_1 is also acceptable. For example, we can write the query as the rule:

$$Q(X_2, X_1) \text{ :- } P(X_1, Z_1, Z_2), R(Z_2, Z_3), R(Z_3, X_2).$$

Let Q_1 and Q_2 be two n-ary queries having the same tuple of distinguished variables. If $Q_1(D) \subseteq Q_2(D)$ for every database D, we say that Q_1 is *contained in* Q_2, and write

$Q_1 \subseteq Q_2$. The *conjunctive-query containment* problem asks: given two conjunctive queries Q_1 and Q_2, is $Q_1 \subseteq Q_2$?

It is well known that conjunctive-query containment can be reformulated as a *conjunctive-query evaluation* problem and also as a *homomorphism* problem. The link to these two other problems is via the *canonical* database D^Q associated with Q. This database is defined as follows. Each variable occurring in Q is considered a distinct element in D^Q. Every predicate in the body of Q is a predicate of D^Q as well; moreover, for every distinguished variable X_i of Q, there is a distinct unary predicate P_i (not occurring in Q). As regards the facts of D^Q, every subgoal in the body of Q gives rise to a tuple in the corresponding predicate of D^Q, and if X_i is a distinguished variable of Q then $P_i(X_i)$ is a fact of D^Q. Thus, in the example above, the canonical database consists of the facts $P(X_1, Z_1, Z_2)$, $R(Z_2, Z_3)$, $R(Z_3, X_2)$, $P_1(X_1)$, $P_2(X_2)$. Recall that a *homomorphism* between two relational structures A and B over the same vocabulary is a mapping $h : A \to B$ such that if $(c_1, \ldots, c_k) \in P^A$, then $(h(c_1), \ldots, h(c_k)) \in P^B$, where P is any predicate symbol in the vocabulary, and P^A and P^B are the interpretations of P on A and B. The relationship between conjunctive query containment, conjunctive query evaluation, and homomorphisms is provided by the following theorem.

Theorem 2.1: [CM77] *Let Q_1 and Q_2 be two n-ary conjunctive queries having the same tuple of distinguished variables. Then the following statements are equivalent.*

- $Q_1 \subseteq Q_2$.
- $(X_1, \ldots, X_n) \in Q_2(D^{Q_1})$, *where $(X_1, \ldots, X_n)$ is the tuple of the distinguished variables of Q_1.*
- *There is a homomorphism $h : D^{Q_2} \to D^{Q_1}$.*

Note that every database D gives rise to a Boolean conjunctive query Q^D whose body consists of the conjunction of all facts in D, where we view the elements of the databases as existentially quantified variables. In turn, this makes it possible to show that both conjunctive query evaluation and the existence of homomorphism between two finite relational structures are reducible to conjunctive query containment. In particular, there is a homomorphism $h : A \to B$ if and only if $Q^B \subseteq Q^A$.

Let us now focus on the *constraint-satisfaction* problem. As mentioned earlier, this problem is usually formulated as the question: does there exists an assignment of possible values to given variables, so that certain constraints are satisfied? Instead, we will consider an alternate elegant formulation in terms of homomorphisms. Let $\mathcal{A}$ and $\mathcal{B}$ be two classes of finite relational structures. The *(uniform) constraint-satisfaction problem* $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$ is the following decision problem: given a structure $A \in \mathcal{A}$ and a structure $B \in \mathcal{B}$, is there a homomorphism $h : A \to B$? Note that, by its very definition, each $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$ problem is in NP. We write $\mathrm{CSP}(\mathcal{B})$ for the special uniform case $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$ in which $\mathcal{A}$ is the class of all finite relational structures over the vocabulary of B. If $\mathcal{B}$ consists of a single structure B, then we write $\mathrm{CSP}(\mathcal{A}, B)$ instead of $\mathrm{CSP}(\mathcal{A}, \{B\})$. We refer to such problems as *non-uniform* constraint satisfaction problems, because the inputs are just structures A in $\mathcal{A}$. We also write $\mathrm{CSP}(B)$ for the special non-uniform case $\mathrm{CSP}(\mathcal{A}, B)$ in which $\mathcal{A}$ is the class of all finite relational structures over the vocabulary of B. Note that if B is a *Boolean structure*, i.e., it has $\{0,1\}$ as its universe, then $\mathrm{CSP}(B)$ is a *generalized satisfiability* problem in the sense of Schaefer [Sch78] (see also [GJ79, LO6, page 260]). For example, if $B = (\{0,1\}, \{(1,0,0),(0,1,0),(0,0,1)\}$, then $\mathrm{CSP}(B)$ is equivalent to Positive One-in-Three 3-SAT. Thus, $\mathrm{CSP}(B)$ may very well be an NP-complete problem.

We are interested in identifying classes $\mathcal{A}$ and $\mathcal{B}$ such that $\mathrm{CSP}(\mathcal{A},\mathcal{B})$ is solvable in polynomial time. Such classes give rise to tractable cases of the constraint satisfaction problem and, hence, of the conjunctive query containment problem as well. For the past twenty years, researchers in computational complexity have investigated $\mathrm{CSP}(B)$ problems, and have discovered several polynomial-time cases. As a general rule, however, non-uniform tractable results do not uniformize. Indeed, it is not hard to construct classes $\mathcal{A}$ and $\mathcal{B}$ of finite relational structures such that $\mathrm{CSP}(\mathcal{A},\mathcal{B})$ is NP-complete, but for each $B \in \mathcal{B}$ the non-uniform $\mathrm{CSP}(\mathcal{A}, B)$ problem is solvable in polynomial time. For example, let $\mathcal{K}$ be the class of all finite cliques, and let $\mathcal{G}$ be the class of all finite undirected graphs. It is clear that $\mathrm{CSP}(\mathcal{K},\mathcal{G})$ is NP-complete, since it is equivalent to the CLIQUE problem. For every fixed finite undirected graph G, however, one can determine in a constant number of steps whether G has a clique of size k. This example is not isolated, since other NP-complete problems can be viewed this way. In particular, if $\mathcal{P}$ is the class of all finite paths, then $\mathrm{CSP}(\mathcal{P},\mathcal{G})$ is equivalent to the Hamiltonian Path Problem, whereas for every finite graph G there is a linear-time algorithm for $\mathrm{CSP}(\mathcal{P}, G)$. These negative results notwithstanding, in the sequel we will establish that several interesting non-uniform tractable cases do uniformize and give rise to tractable cases of constraint satisfaction and conjunctive query containment.

3 Boolean Constraint Satisfaction

3.1 Tractable Cases

Schaefer studied the computational complexity of Boolean $\mathrm{CSP}(B)$ problems, for which he established a *dichotomy* [Sch78]. More specifically, he identified six classes of Boolean structures and showed that $\mathrm{CSP}(B)$ is solvable in polynomial time, if B is in one of these classes, but $\mathrm{CSP}(B)$ is NP-complete in all other cases. This classification is in terms of defining formulas. A k-ary Boolean relation R can be viewed as a set of truth assignments on the propositional variables $p_1, \ldots, p_k$. Thus, for each k-ary Boolean relation R there is a propositional formula δ_R over the variables $p_1, \ldots, p_k$ such that $R = models(\delta_R)$. We call δ_R a *defining formula* of R, and we say that R is *definable by* δ_R. Schaefer showed that for a Boolean structure B, we have that $\mathrm{CSP}(B)$ is in PTIME if one of the following six cases holds:

1. each relation in B contains the tuple $\langle 0, \ldots, 0\rangle$,
2. each relation in B contains the tuple $\langle 1, \ldots, 1\rangle$,
3. each relation in B is *Horn* (i.e., definable by a CNF formula with at most one positive literal per clause),

4. each relation in B is *dual Horn* (i.e., definable by a CNF formula with at most one negative literal per clause),

5. each relation in B is *bijunctive* (i.e., definable by a 2-CNF formula),

6. each relation in B is *affine* (i.e., definable by a conjunction of linear equations).[1]

Furthermore, Schaefer established that if B is not in any of these six classes, then CSP(B) is NP-complete.

We say that a Boolean structure B is a *Schaefer structure* if B is in at least one of the above six classes, in which case CSP(B) is solvable in polynomial time. We call the class of all Schaefer structures *Schaefer's class*, denoted $\mathcal{SC}$. Our main result in this section is that CSP($\mathcal{SC}$) is solvable in polynomial time, which means that Schaefer's tractability results completely uniformize. As a first step, we need to show that structures in $\mathcal{SC}$ can be recognized in polynomial time. This follows from results in [DP92, Sch78].

Theorem 3.1: *The class $\mathcal{SC}$ is recognizable in polynomial time.*

Proof: The first two cases are trivially recognizable. Schaefer showed that a Boolean relation R is bijunctive if and only if the following condition holds: if $t_1, t_2, t_3 \in R$, then $(t_1 \vee t_2) \wedge (t_2 \vee t_3) \wedge (t_1 \vee t_3) \in R$ (here Boolean operations are applied to tuples componentwise). In addition, Schaefer showed that a Boolean relation R is affine if and only if the following condition holds: if $t_1, t_2, t_3 \in R$, then $(t_1 \oplus t_2 \oplus t_3) \in R$. Finally, Dechter and Pearl [DP92] showed that a Boolean relation R is Horn (resp., dual Horn) if and only if the following condition holds: if $t_1, t_2 \in R$, then $t_1 \wedge t_2 \in R$ (resp., $t_1 \vee t_2 \in R$).[2] Clearly, each of these conditions can be checked in polynomial time. ∎

We say that a relation R is a *trivial Schaefer relation* if it is covered by the first two cases of Schaefer's classification, and we say that R is a *nontrivial Schaefer relation* if it is covered by the four interesting cases of Schaefer's classification (i.e., Horn, dual Horn, bijunctive, and affine). In the latter cases, the relation R is definable by a formula δ_R with a certain syntactical structure. The next step is to show that, given a nontrivial Schaefer relation R, we can construct a defining formula δ_R in polynomial time.

Theorem 3.2: *There is a polynomial algorithm that constructs for each nontrivial Schaefer relation R a defining formula δ_R.*

Proof: There are four cases to consider. Dechter and Pearl [DP92] showed how to construct δ_R in polynomial time, when R is Horn or dual Horn. It remains to deal with the cases that R is bijunctive or affine.

Let R be a k-ary bijunctive relation. Then there is a 2CNF formula α over the propositional variables $\{p_1, \ldots, p_k\}$ such that $R = models(\alpha)$. If c is a 2-clause over $p_1, \ldots, p_k$, we say that R *satisfies* c, denoted $R \models c$, if $R \subseteq models(c)$. Consider the formula $\delta_R = \bigwedge_{R \models c} c$, where the conjunction ranges over all 2-clauses c over $\{p_1, \ldots, p_k\}$. We

[1]A linear equation is a formula of the form $(p_{i_1} \oplus p_{i_2} \oplus \ldots \oplus p_{i_l}) \leftrightarrow$ **false** or $(p_{i_1} \oplus p_{i_2} \oplus \ldots \oplus p_{i_l}) \leftrightarrow$ **true**.

[2]For precursors of this result see [McK43, Riv74].

claim that $R = models(\delta_R)$ and, consequently, δ_R is a defining formula of R. Clearly, $R \subseteq models(\delta_R)$. Moreover, if c is a conjunct in α, then R satisfies c. Thus c is also a conjunct of δ_R and so $models(\delta_R) \subseteq models(\alpha) = R$, which implies that $R = models(\delta_R)$. Clearly, δ_R can be constructed in time $O(||R|| \cdot k^2)$.

Let R be a k-ary affine relation. Note that every linear formula $(p_{i_1} \oplus p_{i_2} \oplus \ldots \oplus p_{i_l}) \leftrightarrow \mathbf{false}$ (resp., $\leftrightarrow \mathbf{true}$) can be viewed as the equation $p_{i_1} + p_{i_2} + \ldots + p_{i_l} = 0$ (resp., $= 1$) over the Boolean field. Let $R' = \{(\mathbf{t}, 1) \mid \mathbf{t} \in R\}$. Each linear equation satisfied by R corresponds to a Boolean $(k+1)$-vector $\mathbf{a} = (a_1, \ldots, a_{k+1})$ such that $a_1 t_1 + \ldots + a_{k+1} t_{k+1} = 0$, for each $\mathbf{t} = (t_1, \ldots, t_{k+1}) \in R'$. Thus, the set of such vectors $\mathbf{a}$ is the nullspace $N_{R'}$ of R'; that is, the vector space of solutions to the homogeneous linear equation system $R'\mathbf{a} = 0$ over the Boolean field, where R' is viewed as a Boolean $|R| \times (k+1)$ matrix (note that $|R| = |R'|$). By the Fundamental Theorem of Linear Algebra, the dimension of the space $N_{R'}$ is at most $\min(k+1, |R|)$. By Gaussian elimination, we can convert R' to a row-echelon matrix in polynomial time and obtain a basis of $N_{R'}$, whose size is at most $\min(k+1, |R|)$ [KW98]. Each vector $\mathbf{a} = (a_1, \ldots, a_{k+1})$ in the basis corresponds to a linear formula $(p_{i_1} \oplus p_{i_2} \oplus \ldots \oplus p_{i_l}) \leftrightarrow \mathbf{false}$ (or, $\leftrightarrow \mathbf{true}$) that is satisfied by R. We claim that the conjunction δ_R of these formulas constitutes a defining formula of R. Clearly $R \subseteq models(\delta_R)$. Moreover, we already observed that each linear equation e satisfied by R corresponds to a vector $\mathbf{a}_e$ in $N_{R'}$. Thus, $\mathbf{a}_e$ can be obtained as a linear combination of basic vectors; in other words, e is a consequence of δ_R and so $models(\delta_R) \subseteq R$. ∎

We can now prove the main result of this section.

Theorem 3.3: CSP($\mathcal{SC}$) *is solvable in polynomial time.*

Proof: Suppose we are given a pair A, B of relational structures, where $B \in \mathcal{SC}$. We have to determine whether there is a homomorphism from A to B. By Theorem 3.1, we can determine in polynomial time which of the six tractable cases in Schaefer's classification describes B. If B is a trivial Schaefer structure, then there is a homomorphism from A to B, so we can assume that B is a nontrivial Schaefer structure. For each k-ary relation Q in A, let Q' be the corresponding relation in B (i.e., Q and Q' are the interpretations of the same relation symbol). Apply Theorem 3.2 to construct $\delta_{Q'}$ (recall that $\delta_{Q'}$ is a formula over $\{p_1, \ldots, p_k\}$.

We can view each element of A as a propositional variable. For a tuple $\mathbf{t} = (t_1, \ldots, t_k) \in Q$, let $\delta_{Q'}(\mathbf{t})$ be the formula obtained from $\delta_{Q'}$ by substituting t_i for p_i, $1 \leq i \leq k$. Let $\varphi_Q = \bigwedge_{\mathbf{t} \in Q} \delta_{Q'}(\mathbf{t})$. Note that the length of φ_Q is $O(|Q||\delta_{Q'}|)$. Let $\varphi_A = \bigwedge_{Q \in A} \varphi_Q$. We claim that there is a homomorphism from A to B precisely when φ_A is satisfiable. Indeed, suppose that there is a homorphism $h : A \to B$. Consider the truth assignment τ defined by $\tau(t_i) = h(t_i)$, for each element t_i of a tuple $\mathbf{t} \in Q$. Choose a specific tuple $\mathbf{t} \in Q$. As $h(\mathbf{t}) \in Q'$, the truth assignment τ' defined by $\tau'(p_i) = h(t_i)$ satisfies the formula $\delta_{Q'}$, so τ satisfies $\delta_{Q'}(\mathbf{t})$. It follows that τ satisfies φ_Q. Conversely, suppose that the truth assignment τ satisfies φ_Q. Define the homomorphism $h(t_i) = \tau(p_i)$ for each element t_i of a tuple $\mathbf{t} \in Q$. Choose a specific tuple $\mathbf{t} \in Q$. As τ satisfies $\delta_{Q'}(\mathbf{t})$, the truth assignment τ' defined by $\tau'(p_i) = \tau(t_i)$ satisfies δ'_Q. It follows that $h(\mathbf{t}) \in Q'$. Note, however, that δ_A is a conjunction of Horn clauses, dual Horn clauses, 2-clauses, or linear formulas, depending on the type of B. Thus, satisfiability of δ_A can be checked in time that is linear in the length of φ_A in the first three cases [BB79, DG84, Pap94], and cubic in the length of φ_A in the fourth case [Sch78]. ∎

When R is an affine relation, the length of the defining formula δ_R constructed above is bounded by the size of R. In contrast, if R is bijunctive, Horn, or dual Horn, then the length of δ_R is proportional to $O(k^2)$ (where k is the arity of R), which can be quite larger than the size of R. Thus, the complexity of our algorithm is cubic in these cases. It is possible, however, to skip the formula-building stage of our algorithm and design a direct algorithm that essentially tests for satisfiability of φ_A without explicitly constructing it.

Theorem 3.4: *Let $\mathcal{B}$ be the class of Horn, dual Horn, or bijunctive structures. Then* $\mathrm{CSP}(\mathcal{B})$ *is solvable in quadratic time.*

Proof: We first describe the algorithm for the Horn case (an analogous algorithm works for the dual Horn case). Let R be a k-ary Horn relation. Take $[k] = \{1, \ldots, k\}$. For $X \subseteq [k]$ and $j \in [k]$, we say that R satisfies $X \to j$ if $R \subseteq \mathit{models}(\bigwedge_{i \in X} p_i \to p_j)$. To determine if there is a homomorphism from a structure A to a Horn structure B, the algorithm maintains a set One of elements of A that have to be mapped to 1. Initially, One is empty. Let $\mathbf{t}$ be a tuple in a relation Q of A. We define $\mathit{One}(\mathbf{t}) = \{i \mid t_i \in \mathit{One}\}$. The algorithm repeatedly selects a tuple $\mathbf{t}$ in a relation Q of A and then adds t_j to One, if Q' satisfies $\mathit{One}(\mathbf{t}) \to j$, where Q' is the relation in B that corresponds to Q. When One cannot be enlarged anymore, there is a homomorphism from A to B if and only if for each tuple $\mathbf{t}$ in a relation Q of A, there is a tuple $\mathbf{t}'$ in the corresponding relation Q' of B such that $\mathit{One}(\mathbf{t}) \subseteq \mathit{One}(\mathbf{t}')$. To prove this claim note that every element in One clearly has to be mapped to 1. Thus, this condition is necessary. To see that it is also sufficient, consider the homomorphism h such $h(t_i) = 1$, if $t_i \in \mathit{One}$, and $h(t_i) = 0$, if $t_i \notin \mathit{One}$. We claim that $h(\mathbf{t}) \in Q'$ for each $\mathbf{t} \in Q$. Indeed, consider the collection T of all tuples $\mathbf{t}' \in Q'$ such that $\mathit{One}(\mathbf{t}) \subseteq \mathit{One}(\mathbf{t}')$. We know that T is not empty. Let $\mathbf{u} = \bigwedge T$ (i.e., the conjunction of all tuples in T). Since Q is a Horn relation, it is closed under conjunction, so $\mathbf{u} \in T \subseteq Q'$ (see proof of Theorem 3.1). If $\mathit{One}(\mathbf{t}) = \mathit{One}(\mathbf{u})$, we are done. Otherwise, there is some $j \in [k]$ such that $j \in \mathit{One}(\mathbf{u}) - \mathit{One}(\mathbf{t})$. But then Q' satisfies $\mathit{One}(\mathbf{t}) \to j$, which means that the algorithm would have added t_j to One, in which case we would have $j \in \mathit{One}(\mathbf{t})$ – contradiction.

We now claim that this algorithm can be implemented to run in time $O(||A|| \cdot ||B||)$. A naive implementation would take time $O(||A||^2 \cdot ||B||)$, since One can be extended at most $||A||$ times, and each extension of One takes time $O(||A|| \cdot ||B||)$, as we have to find a tuple $\mathbf{t} \in Q$, requiring an external loop over all tuples of A, and add t_j to One, if Q' satisfies $\mathit{One}(\mathbf{t}) \to j$, requiring an internal loop over all tuples of B. A more efficient implementation would focus on the elements of A that are to be added to One. In the preprocessing stage, we build linked lists that link all occurrences in A of an element a. When a is added to One, we traverse the list for a and process all tuples $\mathbf{t}$ in which a occur. After this, we update $\mathit{One}(\mathbf{t})$ and then, by scanning B, we check whether this triggers the addition of another element to One. Thus, every occurrence of an element of A is visited at most once, resulting in a running time of $O(||A|| \cdot ||B||)$. (This implementation is inspired by the linear-time algorithms for Horn satisfiability [BB79, DG84].)

Consider now the bijunctive case. A linear-time algorithm for 2-CNF formulas proceeds in phases [LP97]. In each phase, we choose an unassigned variable u and assign an arbitrary truth value to it. We then use the binary clauses in the formula to propagate the assignment. If x is assigned 1 and we have a clause $\neg x \vee y$, then y is assigned 1, and if we have a clause $\neg x \vee \neg y$, then y is assigned 0. Similarly, if x is assigned 0 and we have a clause $x \vee y$, then y is assigned 1, and if we have a clause $x \vee \neg y$, then y is assigned 0. If this results in a variable z assigned both 0 and 1, then we undo all assignments of this phase, and we try to assign to u the other truth value. If both attempts fail, then the formula is unsatisfiable. If either the first or the second attempt is successful, then we proceed to the next phase. As each variable is assigned a truth value at most twice, the algorithm is linear.

Given the pair A, B of structures, where B is bijunctive, we can emulate the above algorithm. The variables are the elements of A. The clauses are implied by the structure B (see proofs of Theorems 3.2 and 3.3). The algorithm proceeds in phases. In each phase, we choose an unassigned element a of A and assign to it a value $i \in \{0,1\}$. We then use the structure B to propagate the assignment. Suppose that $a = t_k$ for a tuple $\mathbf{t}$ in a relation Q of A. Let $T_{Q',k,i}$ be the set of all tuples $\mathbf{t}'$ in the corresponding relation Q' of B such that $t'_k = i$. Suppose now that for some $j \in \{0,1\}$ we have that $t'_l = j$ for all $\mathbf{t}' \in T_{Q',k,i}$; in this case, we know that the element t_l must be assigned the value j. If this propagation results in an element b of A assigned both 0 and 1, then we undo all assignments of this phase, and we try to assign the value $1-i$ to a. If both attempts fail, then there is no homomorphism from A to B. If the first or second attempt are successful, then we proceed to the next phase. Note that each element is assigned value at most twice, but propagating a value requires scanning the pairs t'_k, t'_l of all tuples $\mathbf{t}' \in Q'$. Listing components of a tuple without listing the whole tuple, requires preprocessing the structures to construct the appropriate linked lists. Thus, the complexity of our algorithm is $O(||A|| \cdot |B| + ||B||)$. (Note that $||B||$ is the size of the encoding of B, while $|B|$ is the number of tuples in B.) ∎

3.2 Applications

What are the implications of Theorem 3.3 for conjunctive-query containment? At first sight, it seems that its applicability is limited, since Boolean constraint-satisfaction problems correspond to testing whether $Q_1 \subseteq Q_2$, where Q_1 uses only two variables (corresponding to the Boolean values 0 and 1), and thus seems very restricted. Nonetheless, the critical observation is that every instance (A, B) of a constraint-satisfaction problem can be converted, with a small blow-up, to a Boolean constraint-satisfaction problem (A_b, B_b) by encoding all elements of B in binary notation. Specifically, if n is the number of elements in B, then we can encode every element of B by a bit vector of length $m = \lceil \log n \rceil$. Thus, a k-ary relation Q' of B becomes a km-ary Boolean relation Q'_b of B_b. Note that since one needs $n \lceil \log n \rceil$ bits to encode n elements, there is essentially no blow-up in this conversion. We then replace every element a in A by an m-vector $\langle a_1, \ldots, a_m \rangle$ consisting of m distinct copies of a. For each relation Q of A, this yields a km-ary relation Q_b. This conversion blows up the size of the instance by a factor of $\lceil \log n \rceil$, where $n = |B|$.

Lemma 3.5: *There is a homomorphism from A to B if and only if there is a homomorphism from A_b to B_b.*

Proof: We can assume that the elements of B are $1, \ldots, n$. Suppose first that there is a homomorphism $h : A \to B$. For each element a of A, if $h(a) = j$, then define $h_b(a_i)$ to be the i-th bit of j, for $i = 1, \ldots, m$. It is easy to see that h_b is a homomorphism from A_b to B_b. Suppose now that there is a homomorphism $h_b : A_b \to B_b$. For each element a of A, define $h(a)$ to be the number whose binary notation is $\langle h_b(a_1), \ldots, h_b(a_m)\rangle$. It is easy to see that h is a homomorphism from A to B. ∎

We refer to the process of converting a constraint-satisfaction problem to a Boolean constraint-satisfaction problem as *Booleanization.*

We now present an application of this technique. A *two-atom* conjunctive query is one in which every database predicate occurs at most twice in the body.

Proposition 3.6: [Sar91] *Testing whether a two-atom conjunctive query Q_1 is contained in a conjunctive query Q_2 can be done in polynomial time.*

Proof: By Lemma 3.5, we can Booleanize the problem and reduce it to testing the existence of a homomorphism from a structure A to a Boolean structure B, where every relation in B has at most two tuples. Recall that if B has n elements, then the conversion increases the arity of the relations in A by a factor of $\lceil \log n \rceil$. By the criterion for bijunctivity (see the proof of Theorem 3.1, every relation in B is indeed bijunctive. By Theorem 3.3 and Theorem 3.4, the test can be done in time $O((||Q_2|| \cdot \log ||Q_1||) + ||Q_1||)$. ∎

It is worth noting that the proof in [Sar91] yields a slightly better upper bound, as it is shown there that testing whether a two-atom conjunctive query Q_1 is contained in a conjunctive query Q_2 can be done in time $O(||Q_1|| + ||Q_2||)$.

We conclude this section by presenting two examples that provide additional evidence for the power of Booleanization.

Example 3.7: 2-*Colorability*

Let B be a graph consisting of two nodes and a single undirected edge between them. It is easy to see that CSP(B) is the class of all 2-colorable graphs, and thus a tractable constraint-satisfaction problem. We now show that this well known tractability result can be derived via Booleanization. Indeed, B gives rise to the Boolean structure $B' = (\{0, 1\}, R)$, where $R = \{(0, 1), (1, 0)\}$. This structure is both bijunctive (since R has cardinality 2) and affine (since R is the set of solutions of $(x \oplus y) \leftrightarrow \text{true}$). Thus, Booleanization provides two different explanations as to why 2-Colorability is solvable in polynomial time. ∎

Example 3.8: CSP(C_4)

Let C_4 be a directed cycle with four nodes, that is $C_4 = (\{a, b, c, d\}, E)$, where $E = \{(a, b), (b, c), (c, d), (d, a)\}$. If we Booleanize C_4 using the labeling

$$a \mapsto 00,\ b \mapsto 01,\ c \mapsto 10,\ d \mapsto 11,$$

then we obtain the Boolean structure $C_4' = (\{0, 1\}, E')$, where

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

Clearly, E' is neither 0-valid nor 1-valid. Using the criteria in the proof of Theorem 3.1, it can be easily verified that E' is not Horn, dual Horn or bijunctive, but it is an affine Boolean relation. For instance, E' is not Horn (resp. dual Horn), because the componentwise $\wedge$ (resp. $\vee$) of the first two tuples of E' is $(0,0,0,0)$ (resp. $(0,1,1,1)$), which is not in E'. Similarly, E' is not bijunctive, because the componentwise majority of the first three tuples of E' is $(0,0,1,1)$, which is not in E'. Finally, E' is affine, because it is closed by taking the componentwise $\oplus$ of arbitrary triples in E'. Alternatively, E' can be seen to be affine by observing that E' is the set of solutions of the system

$$(x \oplus y \oplus z) \leftrightarrow \text{false}, \ (y \oplus w) \leftrightarrow \textbf{true}.$$

It follows that $\mathrm{CSP}(C_4)$ is solvable in polynomial time. Naturally, this could also have been seen directly by observing that $\mathrm{CSP}(C_4)$ is 2-Colorability in disguise. Indeed, since homomorphisms compose and since C_4 is 2-colorable, it is easy to see that there is a homomorphism from a given a directed graph G to C_4 if and only if G is 2-colorable [HN90]. ∎

It should be pointed out that the way Booleanization is carried out may give rise to a Schaefer structure of different type. Specifically, we claim that there is a labeling of C_4 that results into a Boolean structure that is both affine and bijunctive. To see this, consider the labeling

$$a \mapsto 00, \ b \mapsto 10, \ c \mapsto 11, \ d \mapsto 01.$$

The resulting Boolean structure is $B'' = (\{0,1\}, E'')$, where

$$E'' = \{(0,0,1,0), (1,0,1,1), (1,1,0,1), (0,1,0,0).$$

We leave it as an exercise for the reader to verify, using the criteria in the proof of Theorem 3.1, that E'' is neither Horn nor dual Horn, but it is both bijunctive and affine.

4 Datalog and Constraint Satisfaction

4.1 Datalog and Finite-Variable Logics

A Datalog program is a finite set of rules of the form

$$t_0 \text{ :- } t_1, \ldots, t_m,$$

where each t_i is an atomic formula $R(x_1, \ldots, x_n)$. The relational predicates that occur in the heads of the rules are the *intensional database* predicates (IDBs), while all others are the *extensional database* predicates (EDBs). One of the IDBs is designated as the *goal* of the program. Note that IDBs may occur in the bodies of rules and, thus, a Datalog program is a recursive specification of the IDBs with semantics obtained via least fixed-points of monotone operators (see [Ull89]). Each Datalog program defines a query which, given a set of EDB predicates, returns the value of the goal predicate. Moreover, this query is computable in polynomial time, since the bottom-up evaluation of the least fixed-point of the program terminates within a polynomial number of steps (in the size of the given EDBs) (see [Ull89]). Thus, expressibility in Datalog is a sufficient condition for tractability of a query.

If B is a finite relational structure and $\mathcal{A}$ is a class of structures, then we write $\neg\mathrm{CSP}(\mathcal{A}, B)$ for the *complement* of $\mathrm{CSP}(\mathcal{A}, B)$, that is, the class of structures A such that there is no homomorphism $h : A \to B$. Feder and Vardi [FV99] provided a unifying explanation for the tractability of many non-uniform $\mathrm{CSP}(B)$ problems by showing that the complement of each of these problems is expressible in Datalog. Our aim in this section is to obtain stronger uniform tractability results for the collections of constraint satisfaction problems whose complements are expressible in Datalog with a bounded number of distinct variables.

For every positive integer k, let k-Datalog be the collection of all Datalog programs in which the body of every rule has at most k distinct variables and also the head of every rule has at most k variables (the variables of the body may be different from the variables of the head). For example, the query Non-2-Colorability is expressible in 4-Datalog, since it is definable by the goal predicate Q of the following Datalog program, which asserts that a cycle of odd length exists:

$$
\begin{array}{rcl}
P(X,Y) & :- & E(X,Y) \\
P(X,Y) & :- & P(X,Z), E(Z,W), E(W,Y) \\
Q & :- & P(X,X).
\end{array}
$$

It is well known that Datalog can be viewed as a fragment of least fixed-point logic LFP (see [CH85, AHV95]). In turn, on the class of all finite structures LFP is subsumed by the finite-variable infinitary logic $\mathcal{L}^{\omega}_{\infty\omega} = \bigcup_k \mathcal{L}^{k}_{\infty\omega}$, where $\mathcal{L}^{k}_{\infty\omega}$ is the infinitary logic with arbitrary disjunctions and conjunctions, but with at most k distinct variables (see [KV92]). In the present paper, we are interested in fragments of $\mathcal{L}^{k}_{\infty\omega}$ and $\mathcal{L}^{\omega}_{\infty\omega}$ that are suitable for the study of Datalog. For every $k \geq 1$, let $\exists\mathcal{L}^{k}_{\infty\omega}$ be the *existential positive* fragment of $\mathcal{L}^{\omega}_{\infty\omega}$ with k variables, that is, the collection of all formulas that have at most k distinct variables and are obtained from atomic formulas using infinitary disjunction, infinitary conjunction, and existential quantification only. Let Q be a query on the class of all finite structures over a fixed vocabulary σ. In [KV95], it was shown that if Q is expressible in k-Datalog, then Q is also definable in $\exists\mathcal{L}^{k'}_{\infty\omega}$ for some $k' > k$. Moreover, in [KV96] it was shown that if Q is expressible in LFP^k (least fixed-point logic with k variables), then Q is also expressible in $\mathcal{L}^{k}_{\infty\omega}$. As a matter of fact, the proof can be adapted to yield the following result, which is optimal as regards the number of distinct variables used.

Theorem 4.1: *Let k be a positive integer. Every k-*Datalog *query is expressible in $\exists\mathcal{L}^{k}_{\infty\omega}$. Thus, k-*Datalog $\subseteq \exists\mathcal{L}^{k}_{\infty\omega}$.

In what follows, we present a self-contained proof of the preceding Theorem 4.1. For this, we first have to give precise definitions of the concepts involved and establish a number of intermediate results.

Let σ be a fixed relational vocabulary. For every $k \geq 1$, we write FO^k for the collection of all first-order formulas with at most k distinct variables. We also write $\exists\mathrm{FO}^k$ for the existential positive fragment of FO^k, i.e., the collection of all first-order formulas that have at most k distinct variables and are obtained from atomic formulas using disjunction, conjunction, and existential quantification only.

A *system* of first-order formulas is a finite sequence

$$\varphi_1(x_1,\ldots,x_{n_1},S_1,\ldots,S_l),\ldots,\varphi_l(x_1,\ldots,x_{n_l},S_1,\ldots,S_l)$$

of first-order formulas such that each S_i is a relation symbol of arity n_i, $1 \le i \le l$, not in the vocabulary σ. If A is a σ-structure, then every such system gives rise to an operator Φ from sequences $(R_1,\ldots,R_l)$ of relations R_i of arity n_i, $1 \le i \le l$, on the universe $\mathbf{A}$ to sequences of relations on the universe of A of the same arities. More precisely,

$$\Phi(R_1,\ldots,R_l) = (\Phi_1(R_1,\ldots,R_l),\ldots,\Phi_l(R_1,\ldots,R_l)),$$

where for every $i \le l$

$$\Phi_i(R_1,\ldots,R_l) = \{(a_1,\ldots,a_{n_i}) : A \models \varphi_i(x_1/a_1,\ldots,x_{n_i}/a_{n_i},S_1/R_1,\ldots,S_l/R_l)\}.$$

The *stages* $\Phi^m = (\Phi_1^m,\ldots,\Phi_l^m)$, $m \ge 1$, of Φ on a σ-structure A are defined by the following induction on m simultaneously for all $i \le l$:

$$\Phi_i^1 = \Phi_i(\emptyset,\ldots,\emptyset),\ \Phi_i^{m+1} = \Phi_i(\Phi_1^m,\ldots,\Phi_l^m),\ i \le l,\ m \ge 1.$$

If each formula $\varphi_i(x_1,\ldots,x_{n_i},S_1,\ldots,S_l)$, $1 \le i \le l$, of a system is positive in the relation symbols $S_1,\ldots,S_l$, then the associated operator Φ is monotone in each of its arguments and, as a result, the sequence of its stages is increasing in each component. Thus, for every finite structure A the sequence of stages of Φ converges after finitely many iterations, i.e., there is a positive integer m_0 such that $\Phi^m = \Phi^{m_0}$ for every $m \ge m_0$. Moreover, the sequence $\Phi^{m_0} = (\Phi_1^{m_0},\ldots,\Phi_l^{m_0})$ is the least fixed-point of the operator Φ on A, i.e., the smallest sequence $(R_1,\ldots,R_l)$ of relations on A such that $\Phi(R_1,\ldots,R_l) = (R_1,\ldots,R_l)$ (see [AHV95]). We call this sequence the *least fixed-point of the system* $\varphi_1,\ldots,\varphi_l$ and denote it by $(\varphi_1^\infty,\ldots,\varphi_l^\infty)$. Usually, one is interested not in the entire sequence $(\varphi_1^\infty,\ldots,\varphi_l^\infty)$, but in only one of its components; for instance, in the last component φ_l^∞.

Least fixed-point logic LFP is the extension of first-order logic that has as formulas the components φ_i^∞ of systems $\varphi_1,\ldots,\varphi_l$ of positive first-order formulas. For every $k \ge 1$, let LFP^k be the fragment of LFP obtained by taking the components of least fixed-points of systems of positive FO^k-formulas. Similarly, $\exists\mathrm{LFP}^k$ is the fragment of LFP obtained by taking the componets of least fixed-points of systems of positive $\exists\mathrm{FO}^k$-formulas.

Chandra and Harel [CH85] showed that Datalog has the same expressive power as the existential fragment of LFP. More precisely, a query is expressible in k-Datalog if and only if it is $\exists\mathrm{LFP}^k$-definable. In fact, every k-Datalog program ρ can be "simulated" by a system of positive $\exists\mathrm{FO}^k$-formulas, and vice versa. Intuitively, every IDB predicate P of ρ gives rise to an $\exists\mathrm{FO}^k$-formula that is the disjunction of the positive existential formulas that define the bodies of the rules having the IDB predicate P as head. The resulting system of $\exists\mathrm{FO}^k$-formulas simulates the k-Datalog program ρ "step-by-step", that is to say, each stage of the system corresponds to a stage in the "bottom-up" evaluation of ρ. Consequently, to prove Theorem 4.1 suffices to establish that $\exists\mathrm{LFP}^k \subseteq \exists\mathcal{L}^k_{\infty\omega}$, which amounts to establishing that if $\varphi_1,\ldots,\varphi_l$ is a system of positive $\exists\mathrm{FO}^k$-formulas, then each component φ_i^∞ of the least fixed-point of this system is $\exists\mathcal{L}^k_{\infty\omega}$-definable.

In the sequel, we assume that for every $k \ge 1$ the variables $x_1,\ldots,x_k$ are the k distinct variables of the logics $\exists\mathrm{FO}^k$ and $\exists\mathcal{L}^k_{\infty\omega}$.

Lemma 4.2: *Let k be a positive integer, let $\pi : \{1, \ldots, k\} \mapsto \{1, \ldots, k\}$ be a function, and let Q be a query.*

- *If Q is $\exists\mathrm{FO}^k$-definable, then the query Q_π is also $\exists\mathrm{FO}^k$-definable, where for every finite σ-structure A and every sequence $(a_1, \ldots, a_k)$ of elements from the universe of A*

$$(a_1, \ldots, a_k) \in Q_\pi(A) \Longleftrightarrow (a_{\pi(1)}, \ldots, a_{\pi(k)}) \in Q(A).$$

- *If Q is $\exists\mathcal{L}^k_{\infty\omega}$-definable, then the query Q_π is also $\exists\mathcal{L}^k_{\infty\omega}$-definable.*

Proof: We will show that for every function $\pi : \{1, \ldots, k\} \mapsto \{1, \ldots, k\}$ and for every formula $\varphi(x_1, \ldots, x_k)$ of $\exists\mathrm{FO}^k$ (resp., $\exists\mathcal{L}^k_{\infty\omega}$) there is a formula $\varphi_\pi(x_1, \ldots, x_k)$ of $\exists\mathrm{FO}^k$ (resp., $\exists\mathcal{L}^k_{\infty\omega}$) such that for every σ-structure A and every sequence $(a_1, \ldots, a_k)$ of elements from the universe of A

$$A \models \varphi_\pi(x_1/a_1, \ldots, x_k/a_k) \Longleftrightarrow A \models \varphi(x_1/a_{\pi(1)}, \ldots, x_k/a_{\pi(k)}).$$

The proof is by induction on the construction of $\exists\mathrm{FO}^k$-formulas (resp., $\exists\mathcal{L}^k_{\infty\omega}$-formulas) simultaneously for all functions π.

- If $\varphi(x_1, \ldots, x_k)$ is the formula $x_i = x_j$ for some i, j with $1 \leq i \leq j \leq k$, then $\varphi_\pi(x_1, \ldots, x_k)$ is the formula $x_{\pi(i)} = x_{\pi(j)}$.

- If $q : \{1, \ldots, r\} \mapsto \{1, \ldots, k\}$ is a function, R is a relation symbol in σ of arity r, and $\varphi(x_1, \ldots, x_k)$ is the atomic formula $R(x_{q(1)}, \ldots, x_{q(r)})$, then $\varphi_\pi(x_1, \ldots, x_k)$ is the formula $R(x_{\pi(q(1))}, \ldots, x_{\pi(q(r))})$.

- If $\varphi_\pi(x_1, \ldots, x_k)$ is of the form $\psi(x_1, \ldots, x_k) \wedge \chi(x_1, \ldots, x_k)$, then $\varphi_\pi(x_1, \ldots, x_k)$ is the formula $\psi_\pi(x_1, \ldots, x_k) \wedge \chi_\pi(x_1, \ldots, x_k)$. Moreover, if $\varphi_\pi(x_1, \ldots, x_k)$ is of the form $\bigwedge_{\psi \in \Psi} \psi(x_1, \ldots, x_k)$, then $\varphi_\pi(x_1, \ldots, x_k)$ is the formula $\bigwedge_{\psi \in \Psi} \psi_\pi(x_1, \ldots, x_k)$. The case of disjunction is handled in a similar manner.

- Finally, assume that $\varphi(x_1, \ldots, x_k)$ is a formula of the form $(\exists x_j)\psi(x_1, \ldots, x_k)$ for some $j \leq k$. There are two cases to consider. Suppose first that there is no j' such that $j' \leq k$, $j' \neq j$, and $\pi(j) = \pi(j')$. Then the desired formula $\varphi_\pi(x_1, \ldots, x_k)$ is the formula $(\exists x_{\pi(j)})\psi_\pi(x_1, \ldots, x_k)$. Suppose on the other hand that there is some j' such that $j' \leq k$, $j' \neq j$, and $\pi(j) = \pi(j')$. Then there is some $j'' \leq k$ such that j'' is not in the range of π. Let $\pi' : \{1, \ldots, k\} \mapsto \{1, \ldots, k\}$ be the function such that $\pi'(i) = \pi(i)$, if $i \neq j$, and $\pi'(j) = j''$. By applying the induction hypothesis to the function π' and to the formula $\psi(x_1, \ldots, x_k)$, we obtain a formula $\psi_{\pi'}$ of $\exists\mathrm{FO}^k$ such that for all σ-structures A and all sequences of elements $(a_1, \ldots, a_k)$ from the universe of A

$$A \models \psi_{\pi'}(x_1/a_1, \ldots, x_k/a_k) \Longleftrightarrow A \models \psi(x_1/a_{\pi'(1)}, \ldots, x_k/a_{\pi'(k)}).$$

  Then the desired formula $\varphi_\pi(x_1, \ldots, x_k)$ is the formula $(\exists x_{j''})\psi_{\pi'}(x_1, \ldots, x_k)$. ∎

We are now ready to show that $\exists\mathrm{LFP}^k \subseteq \exists\mathcal{L}^k_{\infty\omega}$, for every $k \geq 1$, which will imply that k-Datalog $\subseteq \exists\mathcal{L}^k_{\infty\omega}$.

Theorem 4.3: *Let k, $n_1, \ldots, n_l$ be positive integers such that $n_i \leq k$ for every $i \leq l$, let $S_1, \ldots S_l$ be relation symbols not in the vocabulary σ and having arities $n_1, \ldots, n_l$, and let*

$$\varphi_1(x_1, \ldots, x_{n_1}, S_1, \ldots, S_l), \ldots, \varphi_l(x_1, \ldots, x_{n_l}, S_1, \ldots, S_l)$$

be a system of positive $\exists\mathrm{FO}^k$-formulas over the vocabulary $\sigma \cup \{S_1, \ldots, S_l\}$. Then the following statements are true for the above system and for the operator Φ associated with it.

- *For every $m \geq 1$, each component Φ_i^m, $1 \leq i \leq l$, of the stage $\Phi^m = (\Phi_1^m, \ldots, \Phi_l^m)$ is definable by an $\exists\mathrm{FO}^k$-formula on all σ-structures (finite or infinite).*
- *Each component φ_i^∞, $1 \leq i \leq l$, of the least fixed-point $(\varphi_1^\infty, \ldots, \varphi_l^\infty)$ of the system is definable by an $\exists\mathcal{L}_{\infty\omega}^k$-formula on all σ-structures (finite or infinite).*

Proof: Assume first that $n_i = k$ for all $i \leq l$, which means that each S_i is a k-ary relation symbol not in σ and each $\varphi_i(x_1, \ldots, x_k, S_1, \ldots, S_l)$, $1 \leq i \leq k$, is a formula of $\exists\mathrm{FO}^k$ over the vocabulary $\sigma \cup \{S_1, \ldots, S_l\}$. By induction on m simultaneously for all $i \leq l$, we will show that each component Φ_i^m of every stage Φ^m is definable by a formula $\varphi_i^m(x_1, \ldots, x_k)$ of $\exists\mathrm{FO}^k$. The claim is obvious for $m = 1$, since each component Φ_i^1 of the stage Φ^1 is definable by the $\exists\mathrm{FO}^k$ formula $\varphi_i(x_1, \ldots, x_k, S_1/\emptyset, \ldots, S_l/\emptyset)$, $1 \leq i \leq l$. Assume now that there are $\exists\mathrm{FO}^k$-formulas $\varphi_i^m(x_1, \ldots, x_k)$ that define the components Φ_i^m, $1 \leq i \leq m$ of the stage Φ^m, which means that for every structure A and every sequence $(a_1, \ldots, a_k)$ of elements from the universe of A

$$(a_1, \ldots, a_k) \in \Phi_i^m \Longleftrightarrow \models \varphi_i^m(x_1/a_1, \ldots, x_k/a_k),\ 1 \leq i \leq l.$$

Let us consider the components Φ_i^{m+1} of the stage Φ^{m+1}, which are defined by

$$(a_1, \ldots, a_k) \in \Phi_i^{m+1} \Longleftrightarrow A \models \varphi_i(x_1/a_1, \ldots, x_k/a_k, S_1/\Phi_1^m, \ldots, S_l/\Phi_l^m).$$

Every occurrence of each relation symbol S_j, $1 \leq j \leq l$, in the formulas of the system is in a subformula of the form $S_j(x_{\pi(1)}, \ldots, x_{\pi(k)})$ for some function $\pi : \{1, \ldots, k\} \mapsto \{1, \ldots, k\}$. Since each relation symbol $S_1, \ldots, S_l$ has only positive occurrences in the formulas of the system, by using the induction hypothesis and repeatedly applying Lemma 4.2, for each $j \leq l$ and each such function π we obtain a formula $\varphi_{j,\pi}^m(x_1, \ldots, x_k)$ of $\exists\mathrm{FO}^k$ such that

$$(a_{\pi(1)}, \ldots, a_{\pi(k)}) \in \Phi_j^m \Longleftrightarrow A \models \varphi_{j,\pi}(x_1/a_1, \ldots, x_k/a_k).$$

For $i \leq l$, let $\varphi_i^{m+1}(x_1, \ldots, x_k)$ be the formula obtained from $\varphi_i(x_1, \ldots, x_k, S_1, \ldots, S_l)$ by substituting each subformula $S_j(x_{\pi(1)}, \ldots, x_{\pi(k)})$ by the corresponding formula $\varphi_{j,\pi}^m(x_1, \ldots, x_k)$. Note that we are using the formulas $\varphi_{j,\pi}^m(x_1, \ldots, x_k)$ instead of the formula $\varphi_j^m(x_1, \ldots, x_k)$, so that these substitutions can be carried out without renaming variables or introducing new variables. Thus, for every $i \leq m$ the formula $\varphi_i^{m+1}(x_1, \ldots, x_k)$ is an $\exists\mathrm{FO}^k$-formula that defines the component Φ_i^m of the stage Φ^{m+1}.

Consider next the case that $n_i < k$ for at least one $i \leq l$. For every $i \leq l$, let T_i be a k-ary relation symbol not in the vocabulary σ and let $\psi_i(x_1, \ldots, x_{n_i}, x_{n_i+1}, \ldots, x_k, T_1, \ldots, T_l)$ be the

$\exists\mathrm{FO}^k$-formula over the vocabulary $\sigma \cup \{T_1, \ldots, T_l\}$ obtained from $\varphi_i(x_1, \ldots, x_{n_i}, S_1, \ldots, S_l)$ as follows: if $n_j < k$, then we replace each subformula $S_j(x_{\pi(1)}, \ldots, x_{\pi(n_j)})$ by the formula

$$(\exists x_{n_j+1}) \ldots (\exists x_k) T_j(x_{\pi(1)}, \ldots, x_{\pi(n_j)}, x_{n_j+1}, \ldots, x_k),$$

while if $n_j = k$, then we replace $S_j(x_{\pi(1)}, \ldots, x_{\pi(n_j)})$ by $T_j(x_{\pi(1)}, \ldots, x_{\pi(n_j)})$. A straightforward induction on m simultaneously for all $i \leq m$ shows that if $n_i = k$, then $\Phi_i^m = \Psi_i^m$, while if $n_i < k$, then for every structure A and every sequence $(a_1, \ldots, a_k)$ of elements from the universe of the structure

$$(a_1, \ldots, a_{n_i}) \in \Phi_i^m \iff A \models (\exists a_{n_i+1} \ldots \exists a_k)((a_1, \ldots, a_{n_i}, a_{n_i+1}, \ldots, a_k) \in \Psi_i^m).$$

For every $m \geq 1$ and $i \leq l$, let $\psi_i^m(x_1, \ldots, x_k)$ be the $\exists\mathrm{FO}^k$-formula that defines the component Ψ_i^m of the stage Ψ^m. If $n_i < k$, then we let $\varphi_i^m(x_1, \ldots, x_{n_i})$ be the formula

$$(\exists x_{n_i+1} \ldots \exists x_k) \psi_i^m(x_1, \ldots, x_{n_i}, x_{n_i+1}, \ldots, x_k),$$

while if $n_i = k$, then we let $\varphi_i^m(x_1, \ldots, x_k)$ be the formula $\psi_i^m(x_1, \ldots, x_k)$. Thus $\varphi_i^m(x_1, \ldots, x_{n_i})$ is an $\exists\mathrm{FO}^k$-formula that defines the component Φ_i^m of the stage Φ^m, $1 \leq i \leq l$.

Finally, each component $\varphi_i^\infty(x_1, \ldots, x_{n_i})$ of the least fixed-poin of the system $\varphi_1, \ldots, \varphi_l$ is definable on all σ-structures by the $\exists\mathcal{L}^k_{\infty\omega}$-formula $\bigvee_{m=1}^\infty \varphi_i^m(x_1, \ldots, x_{n_i})$. ∎

As explained earlier, Theorem 4.1 follows immediately from the preceding Theorem 4.3.

Corollary 4.4: *Let k be a positive integer. Then $\exists\mathrm{LFP}^k \subseteq \exists\mathcal{L}^k_{\infty\omega}$ and, consequently, k-Datalog $\subseteq \exists\mathcal{L}^k_{\infty\omega}$.*

It should be pointed out that on the class of all finite structures k-Datalog is properly contained in $\exists\mathcal{L}^k_{\infty\omega}$, since the latter can express non-computable queries.

4.2 Datalog, Pebble Games, and Constraint Satisfaction

Next, we describe certain combinatorial games that will play an important role in the sequel. Let A and B be two relational structures over a common relational vocabulary σ. The *existential k-pebble game on A and B* is played between two players, the *Spoiler* and the *Duplicator*. The Spoiler places k pebbles (one at a time) on elements of A; after each move of the Spoiler, the Duplicator responds by placing a pebble on an element of B. Once all pebbles have been placed, the Spoiler wins if one of the following two conditions holds for the elements a_i and b_i, $1 \leq i \leq k$, of A and B that have been pebbled in the i-th move of the Spoiler and the Duplicator:

1. the correspondence $a_i \mapsto b_i$, $1 \leq i \leq k$, is not a mapping (that is to say, there exists i_1 and i_2 such that $i_1 \neq i_2$, $a_{i_1} = a_{i_2}$, and $b_{i_1} \neq b_{i_2}$);

2. the correspondence $a_i \mapsto b_i$, $1 \leq i \leq k$, is a mapping, but it is not a a homomorphism from the substructure of A with universe $\{a_1, \ldots, a_k\}$ to the substructure of B with universe $\{b_1, \ldots, b_k\}$.

If neither of the above two conditions holds, then the Spoiler removes one or more pebbles and the game resumes. We say that the *Duplicator wins the existential k-pebble game on A and B* if he has a strategy that allows him to continue playing "forever", that is, the Spoiler can never win a round of the game. A more formal definition of this concept can be given using families of partial homomorphisms with the *forth property up to k* (see [KV95] for details).

If $(a_1, \ldots, a_k)$ is a k-tuple of elements of A and $(b_1, \ldots, b_k)$ is a k-tuple of elements of B, then we say that the *Duplicator wins the existential k-pebble game on $(A, a_1, \ldots, a_k)$ and $(B, b_1, \ldots, b_k)$*, if $(a_1, \ldots, a_k)$ and $(b_1, \ldots, b_k)$ is a winning configuration for the Duplicator, that is, the Duplicator can win the game if the i-th pebble of the Spoiler has been placed on a_i and the i-th pebble of the Duplicator has been placed on b_i, $1 \leq i \leq k$. The following result from [KV95] shows that expressibility in $\exists\mathcal{L}^k_{\infty\omega}$ can be characterized in terms of the existential k-pebble games.

Theorem 4.5: *Let k be a positive integer and Q a k-ary query on a class $\mathcal{C}$ of finite structures. Then the following two statements are equivalent:*

1. *Q is expressible in $\exists\mathcal{L}^k_{\infty\omega}$ on $\mathcal{C}$.*
2. *If A, B are two structures in $\mathcal{C}$ and $(a_1, \ldots, a_k)$, $(b_1, \ldots, b_k)$ are two k-tuples of elements of A and B such that $A \models Q(a_1, \ldots, a_k)$ and the Duplicator wins the existential k-pebble game on $(A, a_1, \ldots, a_k)$ and $(B, b_1, \ldots, b_k)$, then $B \models Q(b_1, \ldots, b_k)$.*

Corollary 4.6: *Let k be a positive integer and Q a Boolean query on a class $\mathcal{C}$ of finite structures. Then the following two statements are equivalent:*

1. *Q is expressible in $\exists\mathcal{L}^k_{\infty\omega}$ on $\mathcal{C}$.*
2. *If A and B are two structures in $\mathcal{C}$ such that $A \models Q$ and the Duplicator wins the existential k-pebble game on A and B, then $B \models Q$.*

Let σ_1 and σ_2 be two disjoint copies of the vocabulary σ, that is, for each relation symbol R of σ and for $i = 1, 2$, the vocabulary σ_i contains a relation symbol R_i of the same arity as R. We write $\sigma_1 + \sigma_2$ for the vocabulary $\sigma_1 \cup \sigma_2 \cup \{D_1, D_2\}$, where D_1 and D_2 are two new unary relation symbols. Using the vocabulary $\sigma_1 + \sigma_2$, we can encode a pair (A, B) of two σ-structures A and B by a single $\sigma_1 + \sigma_2$-structure $A + B$ defined as follows: the universe of $A + B$ is the union of the universes of A and B, the interpretation of D_1 (respectively, D_2) is the universe of A (respectively, B), and the interpretation of each relation symbol R_1 (respectively, R_2) is the interpretation of the relation symbol R on A (respectively, on B). This encoding makes it possible to formally view queries on pairs of σ-structures as queries on single $\sigma_1 + \sigma_2$-structures.

Our next result concerns the computational and descriptive complexity of existential k-pebble games.

Theorem 4.7: *Let σ be a relational vocabulary and let k be a positive integer.*

1. *There is a sentence ψ of least fixed-point logic* LFP *over the vocabulary $\sigma_1 + \sigma_2$ that expresses the query: "Given two σ-structures A and B, does the Spoiler win the existential k-pebble on A and B?".*

   As a result, there is a polynomial-time algorithm such that, given two finite σ-structures A and B, it determines whether the Spoiler wins the existential k-pebble game on A and B.

2. *For every finite σ-structure B, there is a k-*Datalog *program ρ_B that expresses the query "Given a σ-structure A, does the Spoiler win the existential k-pebble game on A and B?".*

Proof: Let $\theta(x_1, \ldots, x_k, y_1, \ldots, y_k)$ be a quantifier-free formula over the vocabulary $\sigma_1 + \sigma_2$ asserting that the correspondence $x_i \mapsto y_i$, $1 \leq i \leq k$, is not a mapping or it is a mapping that is *not* a homomorphism from the substructure induced by $x_1, \ldots, x_k$ over the vocabulary σ_1 to the substructure induced by $y_1, \ldots, y_k$ over the vocabulary σ_2. Specifically, θ is the disjunction of the following formulas:

- $x_i = x_j \wedge y_i \neq y_j$, for every $i, j \leq k$ such that $i \neq j$.
- $R_1(x_{i_1}, \ldots, x_{i_m}) \wedge \neg R_2(y_{i_1}, \ldots, y_{i_m})$, for every m-ary relation symbol R in σ and every m-ary tuple $(i_1, \ldots, i_m)$ of indices from the set $\{1, \ldots, k\}$.

Let T be a 2k-ary relation symbol not in $\sigma_1 + \sigma_2$ and let $\varphi(x_1, \ldots, x_k, y_1, \ldots, y_k, T)$ be the following positive first-order formula over the vocabulary $\sigma_1 + \sigma_2 \cup \{T\}$:

$$\theta(x_1, \ldots, x_k, y_1, \ldots, y_k) \vee \bigvee_{j=1}^{k} (\exists x_j)(\forall y_j)(D_1(x_j) \wedge (D_2(y_j) \rightarrow T(x_1, \ldots, x_k, y_1, \ldots, y_k))).$$

It is easy to verify that if A, B are σ-structures and $(a_1, \ldots, a_k)$, $(b_1, \ldots, b_k)$ are k-tuples of elements of A and B respectively, then the following statements are equivalent:

1. $A + B \models \varphi^\infty(a_1, \ldots, a_k, b_1, \ldots, b_k)$.
2. The Spoiler wins the existential k-pebble game on $(A, a_1, \ldots, a_k)$ and $(B, b_1, \ldots, b_k)$.

Let ψ be the sentence $(\exists x_1) \cdots (\exists x_k)(\forall y_1) \cdots (\forall y_k)\varphi^\infty(x_1, \ldots, x_k, y_1, \ldots, y_k)$ of least fixed-point logic LFP. Consequently, for every σ-structure A and every σ-structure B the following statements are equivalent:

1. $A + B \models \psi$.
2. The Spoiler wins the existential k-pebble game on A and B.

Since every LFP-expressible query is computable in polynomial time, it follows that there is a polynomial time algorithm that, given two finite σ-structures A and B, tells whether the Spoiler wins the existential k-pebble game on A and B.

Note that the positive first-order formula φ above involves existential quantifiers that are interpreted over the elements of A, and universal quantifiers that are interpreted over the elements of B. Consequently, if B is a fixed finite σ-structure, then the universal quantifiers can be replaced by finitary conjunctions over the elements of B and, thus, φ can be transformed to a k-Datalog program ρ_p that expresses the query: "Given a finite σ-structure A, does the Spoiler win the existential k-pebble game on A and B?". In what follows, we describe this Datalog program in some detail. The goal of ρ_B is a 0-ary predicate S. Let $\mathbf{b} = (b_1, \ldots, b_k)$ be a k-tuple of elements of B. For each such k-tuple, we introduce a k-ary relation symbol $T_{\mathbf{b}}$ and the following rules:

- For every i and j such that $b_i \neq b_j$, we have a rule

  $$T_{\mathbf{b}}(x'_1, \ldots, x'_k) \ :- \qquad ,$$

  where $x'_i = x'_j = x_i$, and $x'_s = x_s$, for $s \neq i, j$.

- For every m-ary relation symbol R of σ and every m-ary tuple $(i_1, \ldots, i_m)$ such that

  $$B, b_{i_1}, \ldots, b_{i_m} \models \neg R(x_{i_1}, \ldots, x_{i_m}),$$

  we have a rule

  $$T_{\mathbf{b}}(x_1, \ldots, x_k) \ :- \ R(x_{i_1}, \ldots, x_{i_m}).$$

- For every j with $1 \leq j \leq k$, we have a rule

  $$T(x_1, \ldots, x_k) \ :- \ \bigwedge_{c \in B} T_{\mathbf{b}[j/c]}(x_1, \ldots, x_{j-1}, y, x_{j+1}, \ldots, x_k),$$

  where $\mathbf{b}[j/c] = (b_1, \ldots, b_{j-1}, c, b_{j+1}, \ldots, b_k)$ and y is a new variable (note, however, that the body of the rule has k variables).

- For the goal predicate S, we have the rule

  $$S \ :- \ \bigwedge_{\mathbf{b} \in B^k} T_{\mathbf{b}}(x_1, \ldots, x_k).$$

∎

We now have all the necessary notation and machinery to establish the main results of this section. The next theorem establishes a connection between expressibility of $\neg\mathrm{CSP}(\mathcal{A}, B)$ in k-Datalog and existential k-pebble game. (A closely related, but somewhat less precise, such connection was established in [FV99]).

Theorem 4.8: *Let k be a positive integer, B a finite relational structure, and $\mathcal{A}$ a class of finite relational structures such that $B \in \mathcal{A}$. Then the following statements are equivalent.*

1. $\neg\mathrm{CSP}(\mathcal{A}, B)$ *is expressible in* k-Datalog *on* $\mathcal{A}$.

2. $\neg\mathrm{CSP}(\mathcal{A}, B)$ *is expressible in* $\exists\mathcal{L}^k_{\infty\omega}$ *on* $\mathcal{A}$.

3. $\neg\mathrm{CSP}(\mathcal{A}, B) = \{A \in \mathcal{A} :$ *The Spoiler wins the existential k-pebble game on A and B*$\}$.

Proof: The implication (1) $\Longrightarrow$ (2) follows from Theorem 4.1. To show that (2) $\Longrightarrow$ (3), assume that ψ is an $\exists\mathcal{L}^k_{\infty\omega}$ sentence that defines $\neg$CSP$(\mathcal{A}, B)$ on $\mathcal{A}$. Let A be a finite relational structure in $\mathcal{A}$. If $A \notin$ CSP$(\mathcal{A}, B)$, then $A \models \psi$ and, hence, the Spoiler wins the existential k-pebble game on A and B. Indeed, if the Duplicator wins this game, then, by Theorem 4.6, $B \models \psi$, which means that there is no homomorphism from B to B, a contradiction. Conversely, if the Spoiler wins the existential k-pebble game on A and B, then $A \in \neg$CSP$(\mathcal{A}, B)$. Indeed, otherwise, there is a homomorphism $h : A \to B$, which will give the Duplicator a winning strategy for the existential k-pebble game on A and B; whenever the Spoiler places a pebble on an element a of A, the Duplicator responds by placing the corresponding pebble on the element $h(a)$ of B. Finally, the implication (3) $\Longrightarrow$ (1) follows from Theorem 4.7. ∎

By combining Theorems 4.7 and 4.8, we obtain the following uniform tractability result for classes of constraint satisfaction problems expressible in Datalog.

Theorem 4.9: *Let k be a positive integer, $\mathcal{A}$ a class of finite relational structures, and*

$$\mathcal{B} \;=\; \{B \in \mathcal{A} : \neg\mathrm{CSP}(\mathcal{A}, B) \textit{ is expressible in } k\text{-Datalog}\}.$$

Then the uniform constraint satisfaction problem CSP$(\mathcal{A}, \mathcal{B})$ *is solvable in polynomial time. Moreover, the running time of the algorithm is $O(n^{2k})$, where n is the maximum of the sizes of the input structures A and B.*

We note that it is an open problem whether the class

$$\{B : \neg\mathrm{CSP}(\mathcal{A}, B) \text{ is expressible in } k - \text{Datalog}\}$$

is recursive. In contrast, Schaefer's class $\mathcal{SC}$, which was the basis for the tractability result of Theorem 3.3, is recursive (per Theorem 3.1).

Remark 4.10: Some remarks concerning the results of this section are in order now.

1. Feder and Vardi [FV99] showed that for every non-uniform CSP(B) problem there is a certain k-Datalog program π_B such that if the complement of CSP(B) is expressible in k-Datalog, then π_B expresses it. The preceding Theorems 4.7 and 4.8 give an alternative proof of this result; moreover, they reveal that as π_B we can take the k-Datalog program ρ_B that expresses the query: "Given A, does the Spoiler win the existential k-pebble game on A and B?".

2. To illustrate an application of Theorem 4.9, consider a k-ary Horn Boolean structure B. Then it is easy to verify that the complement of CSP(B) is expressible in k-Datalog. Consequently, Theorem 4.9 yields a polynomial-time algorithm for CSP$(\mathcal{F}, \mathcal{B})$, where $\mathcal{F}$ is the class of all finite structures and $\mathcal{B}$ is the class of k-ary Horn Boolean structures.

∎

5 Bounded Treewidth and Constraint Satisfaction

Up to this point, we found tractable cases of the uniform constraint satisfaction problem CSP($\mathcal{A}, \mathcal{B}$) by imposing restrictions on the class $\mathcal{B}$. In this section, we exhibit tractable cases of CSP($\mathcal{A}, \mathcal{B}$) that are obtained by imposing restrictions on the class $\mathcal{A}$. For this, we consider the concept of *treewidth* of a *relational structure*; this concept was introduced by Feder and Vardi [FV93] and generalizes the concept of treewidth of a graph (see [vL90, Bod93]).

A *tree decomposition* of a finite relational structure A is a labeled tree T such that the following conditions hold:

1. every node of T is labeled by a non-empty subset of the universe V of A,
2. for every relation R of A and every tuple $(a_1, \ldots, a_n)$ in R, there is a node of T whose label contains $\{a_1, \ldots, a_n\}$,
3. for every $a \in V$, the set of nodes X of T whose labels include a forms a subtree of T.

The *width* of a tree decomposition T is the maximum cardinality of a label of a node in T minus 1. Finally, we say that a structure A is of *treewidth* k if k is the smallest positive integer such that A has a tree decomposition of width k.

An alternative way to define the treewidth of a structure A is in terms of the treewidth of the *Gaifman* graph of A [Gai82], that is, the graph that has the elements of the universe of A as nodes and is such that there is an edge between two nodes if and only if the corresponding elements appear in a tuple in one of the relations of A. We call the treewidth of the Gaifman graph of A the *Gaifman treewidth* of A. We now show that the two concepts coincide.

Lemma 5.1: *T is a tree decomposition of a structure A iff it is also a tree decomposition tree of the Gaifman graph of A.*

Proof: It is easy to see that if T is a tree decomposition of A, then T is also a tree decomposition of the Gaifman graph of A. Suppose now that T is a tree decomposition of the Gaifman graph of A. Consider a tuple $(a_1, \ldots, a_n)$ in a relation R of A. The elements $\{a_1, \ldots, a_k\}$ form a clique in the Gaifman graph of A. By Lemma 6.49 of [DF99], there is a node x of T such that $\{a_1, \ldots, a_k\}$ is contained in the label of x. It follows that T is also a tree decomposition of A. ∎

For every $k \geq 1$, let $\mathcal{A}(k)$ be the class of all finite relational structures of treewidth k. Bodlaender [Bod93] showed that, for every $k \geq 1$, there is a linear-time algorithm that tests whether a given graph is of treewidth k. It follows that, for every $k \geq 1$, there is a polynomial-time algorithm that tests whether a given finite relational structure is of treewidth k; in other words, each class $\mathcal{A}(k)$ is recognizable in polynomial time.

Feder and Vardi [FV99] showed for every finite relational structure B, there is a polynomial-time algorithm for the non-uniform constraint satisfaction problem CSP($\mathcal{A}(k), B$). This is also a consequence of the following two facts: for every fixed structure B, the class CSP(B) is known to be expressible in existential monadic second-order logic [FV99]; furthermore, membership in classes of graphs definable in monadic second-order logic is decidable in polynomial time for graphs of bounded-tree width (this is known as Courcelle's Theorem, see

[DF99]). In fact, it has already been shown in [Fre90] that these non-uniform tractability results holds uniformly (see also [DP89]). Here we provide a new proof of uniform tractability via a connection with first-order logic with a bounded number of distinct variables.

Let A and B be two finite relational structure. From Theorem 2.1 and the accompanied remarks, it follows that the existence of a homomorphism $h : A \to B$ is equivalent to whether $Q^A(B)$ is true, where Q^A is the Boolean conjunctive query whose body consist of the conjunction of all facts in A. We show that if A is a finite relational structure of treewidth k, then the conjunctive query Q^A is expressible in $\exists\mathrm{FO}^{k+1}$; moreover, an $\exists\mathrm{FO}^{k+1}$ formula equivalent to Q^A can be found in time polynomial in the size of A.

Lemma 5.2: *Let A be a structure of treewidth k, then Q^A is expressible in $\exists\mathrm{FO}^{k+1}$.*

Proof: The key idea underlying the proof is that structures of bounded treewidth have *parse trees* [DF99] (Chapter 6.4), which can be constructed in polynomial time from tree decompositions. Such parse trees are constructed from *k-boundaried* structures, which are structures with k distinguished nodes labeled $1, \ldots, k$. Such structured can be combined to form larger structures. For example, two k-boundaried structures A and B (we assume a fixed underlying vocabulary) can be *glued* to obtain a structure $A \oplus B$ by taking their disjoint union and then identifying the two nodes labeled i, for $i = 1 \ldots, k$.

A more general way of combining k-boundaried graphs is defined as follows. Let $\alpha = (A, f_1, \ldots, f_n)$ consist of a k-boundaried structure A with domain D and injective mappings $f_i : \{1, \ldots, k\} \to D$. We view α as an n-ary operator of cardinality $|D|$ on k-boundaried structures. Given k-boundaried structures $A_1, \ldots, A_n$, we construct $\alpha(A_1, \ldots, A_n)$ in the following manner. We take the disjoint union of $A, A_1, \ldots, A_n$ and identify the j-th distinguished node of A_i with the node $f_i(j)$ of A, for $i = 1, \ldots, n$, $j = 1, \ldots, k$. The distinguished nodes of the result are the distinguished nodes of A (labels on other nodes are erased). Note that the glue operator $\oplus$ is in essence the binary operator $(\emptyset_k, \iota_k, \iota_k)$, where $\emptyset_k$ is the k-boundaried structure with k elements and empty relations, and ι_k is the identity function on the set $\{1, \ldots, k\}$.

It is shown in [DF99] that there is a polynomial-time algorithm that converts a tree decomposition of a structure C with at least $k+1$ elements of treewidth k to a parse tree (i.e., an expression) in terms of a finite set of unary and binary k-boundaried operators of cardinality k or $k+1$, starting with the constant structure $\emptyset_k$. (Conversely, such a parse tree always describes a structure of treewidth at most k.) A k-boundaried structure A can be viewed as a k-ary conjunctive query Q^A, whose body consists of the conjunction of all facts in A, but where only the non-distinguished variables are existentially quantified, i.e., the distinguished elements are viewed as free variables. We now show by induction that if C is k-boundaried structure expressed as a parse tree of k-boundaried operators of cardinality k or $k+1$ starting with $\emptyset_k$, then Q^C can be expressed in $\exists\mathrm{FO}^{k+1}$. In fact, we prove the stronger claim that Q^C can be expressed in $\exists\mathrm{FO}^{k+1}$ where the tuples of free variables is an arbitrary k-tuple of distinct variables from $\{x_1, \ldots, x_{k+1}\}$.

The claim clearly holds for $\emptyset_k$, whose query is the k-variable conjunctive query with empty body. Consider the expression $\alpha(A_1, \ldots, A_n)$, where $\alpha = (A, f_1, \ldots, f_n)$ is a k-boundaried n-ary operator of cardinality k or $k+1$, and suppose that we have already constructed $\exists\mathrm{FO}^{k+1}$ conjunctive queries $\varphi, \varphi_1, \ldots, \varphi_n$ for the queries $Q^A, Q^{A_1}, \ldots, Q^{A_n}$, respectively. We can take the domain D of Q to be $\{X_1, \ldots, X_k\}$ or $\{X_1, \ldots, X_{k+1}\}$. Recall that f_i is an injective mapping from $\{1, \ldots, k\}$ into D. By the induction hypothesis, we can assume that the tuple of free variabes of φ_i is $(X_{f(1)}, \ldots, X_{f(k)})$. We can also assume that φ is of the form

$$Q(X_{j_1}, \ldots, X_{j_k}) \text{ :- } B,$$

where $X_{j_1}, \ldots, X_{j_k}$ (resp., $X_{j_1}, \ldots, X_{j_k}, X_{j_{k+1}}$) is some permutation of $X_1, \ldots, X_k$ (resp., $X_1, \ldots, X_k, X_{k+1}$). Observe, that there is an implicit existential quantifier when $|D|$ has $k+1$ elements. We can then express $Q^{\alpha(A_1,\ldots,A_n)}$ by

$$Q(X_{j_1}, \ldots, X_{j_k}) \text{ :- } B, \varphi_1, \ldots, \varphi_n.$$

In proof, note that there is a homomorphism from the structure $\alpha(A_1, \ldots, A_n)$ to a structure B iff there are homomorphisms $h : A \to B$, and $h_i : A_i \to B$, for $i = 1, \ldots, n$, such that if a is a distinguished element of A_i labeled j and $f_i(j) = b$, then we must have that $h(b) = h_i(a)$. In other words, to get a homomorphism from $\alpha(A_1, \ldots, A_n)$ to B we need to find homomorphisms from $A, A_1, \ldots, B$ that meet the compatibility conditions required by the mappings $f_1, \ldots, f_n$. The $\exists\mathrm{FO}^{k+1}$ queries $\varphi, \varphi_1, \ldots, \varphi_n$ give us the required $n+1$ homomorphisms, and the compatibility between the different homomorphisms is guaranteed by labeling distinct elements that must be mapped identically by the same variable. ∎

Remark 5.3: The fragment $\exists\mathrm{FO}$ is more general than the fragment of conjunctive queries as it also allows for negations and disjunctions. Consider the fragment $\exists\mathrm{FO}_{\wedge,+}$ that allows no negative formulas and no disjunctions. This fragment has the same expressive power as conjunctive queries. It can be shown that the fragment $\exists\mathrm{FO}^{k+1}_{\wedge,+}$ can express precisely the queries Q^A, where A is a structure of treewidth k. In one direction, note that the translation described in the proof of Lemma 5.2 actually yields a formula in $\exists\mathrm{FO}^{k+1}_{\wedge,+}$. In the other direction, note that the conjunction connective corresponds to the glueing operation $\oplus$ described in the proof of Lemma 5.2, while existential quantification $\exists x_i$ corresponds to the operation of adding a new node, not connected to any of the other nodes, and making it the ith distinguished node; this can be easily expressed as a unary operation of cardinality $k+1$. Thus, given a formula φ in $\exists\mathrm{FO}^{k+1}_{\wedge,+}$, we can construct the parse tree of the structure A by induction on the syntax of φ. Thus, the relationship between treewidth and number of variables is tight. ∎

We can now derive the main result of this section.

Theorem 5.4: *Let k be a positive integer, $\mathcal{A}(k)$ the class of finite relational structures of treewidth k, and $\mathcal{F}$ the class of all finite relational structures. Then the uniform constraint satisfaction problem* $\mathrm{CSP}(\mathcal{A}(k), \mathcal{F})$ *is solvable in polynomial time.*

Proof: We showed in Lemma 5.2 that if A is a finite relational structure of treewidth k, then an $\exists\mathrm{FO}^{k+1}$ formula equivalent to Q^A can be found in time polynomial in the size of A. Thus, in this case, checking the existence of a homomorphism $h : A \to B$ reduces to the evaluation of an $\exists\mathrm{FO}^{k+1}$ query on the structure B. As shown in [Var95], $\exists\mathrm{FO}^{k+1}$ has polynomial-time combined complexity, which implies that $\mathrm{CSP}(\mathcal{A}(k), \mathcal{F})$ is solvable in polynomial time. ∎

A precise complexity analysis of CSP$(\mathcal{A}(k), \mathcal{F})$ is provided in [GLS98], where it is shown that the problem is LOGFCL-complete (LOGCFL is the class of decision problems that are logspace-reducible to a context-free language).

We note that another way to define the treewidth of a structure A is in terms of its *incidence graph* [CR97]. The incidence graph of A is a bipartite graph that has all the tuples in relations of A as nodes in one part, the elements of the universe of A as nodes in the other part, and there is an edge from a node t to a node a iff t is a tuple in A and a is an element that occurs in t. We call the treewidth of the incidence graph of A the *incidence treewidth* of A. Given a tree decomposition T of A, we can convert it into a tree decomposition T' of the incidence graph of A as follows. T' has the same graph structure as T. For every relation R of A and every tuple $t = (a_1, \ldots, a_n)$ in R, there is a node x of T whose label contains $\{a_1, \ldots, a_n\}$. We simply add t (which is a node of the incidence graph of A) to the label of x in T'. It is easy to see that T' is a tree decomposition. Thus, if A has treewidth k, then it has incidence treewidth at most $k + 1$. In the other direction, we can convert a tree decomposition T of the incidence graph of A to a tree decomposition T' of A by replacing a tuple $t = (a_1, \ldots, a_n)$ in the label of a node x of T by the set of elements $\{a_1, \ldots, a_n\}$. Thus, if the incidence treewidth is k, then the treewidth is at most $(k+1)n - 1$, where n is the maximal arity of a relation in A. As an example of the gap between the two notions, consider structure A with a single tuple $(a_1, \ldots, a_n)$. It is easy to see that its treewidth is $n - 1$ (since its Gaifman graph is an n-clique, while its treewidth is 1, since its incidence graph is a tree.

As mentioned in the introduction, Chekuri and Ramajaran [CR98] showed that the uniform constraint satisfaction problem CSP$(\mathcal{Q}(k), \mathcal{F})$ is solvable in polynomial time, where $\mathcal{Q}(k)$ is the class of structures of *querywidth* k (Chekuri and Ramajaran actually studied the conjunctive-query containment problem, which explains the term "querywidth"). (In [CR97] only vocabularies of bounded arities were considered, but the result was extended in [CR98] to general vocabularies.) They also showed that the incidence treewidth of a structure A provides a strict upper bound for its query width by showing that a tree decomposition of the incidence graph is also what they called *query decomposition*. (Note, however, that the property of having treewidth k can be tested in linear time [Bod93], while the property of having querywidth 4 is NP-complete [GLS99].) Thus, the polynomial tractability of CSP$(\mathcal{A}(k), \mathcal{F})$ follows also from the results in [CR98]. Gottlob, Leone, and Scarcello [GLS99] define yet another notion of width, called *hypertree width*. They showed that the querywidth of a structure A provides a strict upper bound for the hypertree width of A, but that the class $\mathcal{H}(k)$ of structures of hypertree width at most k as well as the class CSP$(\mathcal{H}(k), \mathcal{F})$ are efficiently recognizable.

As this discussion shows, the treewidth of a structure A is at least the arity of its widest relation (more precisely, it is at least the number of distinct elements occuring in a tuple of A minus 1). It is desirable, therefore, to decrease the arity of the relations in A. This can be done by encoding the structures A and B by binary structures (i.e., structures with binary relations only). We refer to the binary encoding of a structure A by *binary*(A). The reduction from A to *binary*(A) used here is the *dual-graph representation* of [DP89]. See [BB98] for experimental results concerning this reduction, as well as for another reduction, called the *hidden-variable translation*.

The vocabulary of $binary(A)$ contains a binary relation symbol $E_{P,Q,i,j}$ for each pair of (not necessarily distinct) relations symbols P, Q of A and each pair of argument positions i, j of P, Q, respectively. The domain of $binary(A)$ is the set of tuples occurring in the relations of A. The relation $E_{P,Q,i,j}$ contains the pair (s, t) if the i-th element of s and the j-th element of t are identical. Note that the following three statements are true for the structure $binary(A)$: (a) the relation $E_{P,P,i,i}$ contains all tuples in P; (b) if (s, t) is in the relation $E_{P,Q,i,j}$, then (t, s) is in the relation $E_{Q,P,j,i}$; and (c) if (s, t) is in the relation $E_{P,Q,i,j}$ and (t, u) is in the relation $E_{Q,R,j,k}$, then (s, u) is in the relation $E_{P,R,i,k}$. We refer to this as the reflexive-symmetric-transitive closure of the E relations.

Lemma 5.5: *There is a homomorphism from A to B iff there is a homomorphism from $binary(A)$ to $binary(B)$.*

Proof: Assume that there is homomorphism h from A to B. For a tuple t, define $h(t)$ componentwise. It is easy to see that h is a homomorphism from $binary(A)$ to $binary(B)$: Suppose that $E_{P,Q,i,j}$ contain the pair (s, t) in $binary(A)$, then the elements s_i and t_j are identical, which implies that $h(s_i) = h(t_j)$. It follows that the pair $(h(s), h(t))$ is in the relation $E_{P,Q,i,j}$ in $binary(B)$.

Conversely, suppose that h is a homomorphism from $binary(A)$ to $binary(B)$. Let a be the i-th element of a tuple t in a relation P of A and let b be the i-th element of $h(t)$. We define $h(a)$ to be b. It is easy to see that h is a homomorphism from A to B, provided it is well defined. Suppose a is also the j-th element of a tuple u in a relation Q in A. Then (t, u) is in the relation $E_{P,Q,i,j}$ in $binary(A)$. But then we must have that $(h(t), h(u))$ is in the relation $E_{P,Q,i,j}$ in $binary(B)$, which implies that b is also the j-th element of the tuple $h(u)$. ∎

It is worth noting that in $binary(A)$ it is not necessary to encode all coincidence relations in A. It suffices to put enough tuples there so that the reflexive-symmetric-transitive closure encodes all such coincidence relations. For example, that if (s, t) in the relation $E_{P,Q,i,j}$ and (t, u) is in the relation $E_{Q,R,j,k}$, then it is not necessary to store (s, u) is in the relation $E_{P,R,i,k}$. It is not difficult to prove that Lemma 5.5 still holds. The reason for this optimization is that to minimize the treewidth of $binary(A)$ it is desirable to minimize the number of tuples of $binary(A)$. It is possible to show that, under such an optimized encoding, *acyclic join queries* [AHV95, Ull89] can be encoded by structures of bounded treewidth. While this optimization can be easily done for acyclic instances, it could be very difficult in general. It would be interesting to know the complexity of this task. More formally, consider the following problem OPT: "Let k be a fixed constant. Given a relational structure A, decide whether there exist a suitable encoding of $binary(A)$ (i.e., an encoding that guarantees the coincidence relations, see above) such that its treewidth is at most k." Is OPT decidable in polynomial time?

A class that is more general than the class of bounded treewidth graphs is the class of bounded *cliquewidth* graphs. It is shown in [CO98] than if a graph has treewidth k, then its cliquewidth is bounded from above by $2^{k+1} + 1$. Thus, a class of graphs that has bounded treewidth also has bounded cliquewidth. Courcelle, Makowsky, and Rotics [CMR98] showed that if a class $\mathcal{C}$ is *effectively* of bounded cliquewidth, then every monadic second-order property on $\mathcal{C}$ is polynomial. It follows that CSP$(\mathcal{A}, \mathcal{B})$ is in PTIME for each B if $\mathcal{A}$ is of bounded cliquewidth. On the other hand, every clique has cliquewidth 2 [CO98], and we have observed above that the the class CSP$(\mathcal{K}, \mathcal{G})$ is NP-complete. Thus, while the tractablty result of constraint satisfaction for bounded treewidth structures does uniformize, this is not the case for bounded cliquewidth structures (assuming that P different than NP). It follows that the connection of bounded treewidth to first-order logic with a bounded number of distinct variables does not extend to bounded cliquewidth.

6 Concluding Remarks

In this paper, we brought into center stage the close connection between conjunctive-query containment and constraint satisfaction. Moreover, we showed that several tractable cases of non-uniform CSP(B) problems uniformize and, thus, yield tractable cases of uniform constraint satisfaction and conjunctive query containment.

During the past several years, a group of researchers has pursued tractable cases of constraint satisfaction CSP$(\mathcal{A}, \mathcal{B})$ by investigating the class of functions under which the relations in the structures in $\mathcal{B}$ are closed [JC95, JCG95, JCG96, Jea97] (see [PJ97] for a survey). In [FV93, FV99], a preliminary investigation has been carried out on the connection between expressibility of CSP(B) problems in Datalog and closure of the relations in B under certain functions. In a forthcoming paper, we will elaborate further on this connection and delineate the relationship between the two approaches.

Acknowledgement. We'd like to thank Georg Gottlob, Leonid Libkin, Janos Makowsky, and Werner Nutt for useful discussions. In particular, Georg Gottlob helped us clarify the relationship between treewidth and incidence treewidth in Section 5. We are also grateful to the referees for the useful comments.

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases.* Addison-Wesley, 1995.

[BB79] C. Beeri and P.A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Trans. on Database Systems*, 4:30–59, 1979.

[BB98] F. Bacchus and P. Beek. The conversion between non-binary and binary constraint satisfaction problems, 1998.

[Bib88] W. Bibel. Constraint satisfaction from a deductive viewpoint. *Artificial Intelligence*, 35:401–413, 1988.

[Bod93] H.L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 226–234, 1993.

[CH85] A. Chandra and D. Harel. Horn clause queries and generalizations. *Journal of Logic Programming*, 1:1–15, 1985.

[CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th ACM Symp. on Theory of Computing*, pages 77–90, 1977.

[CMR98] B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on certain structured graph families. In *Graph Theoretic Concepts in Computer Science - WG'98*, Lecture Notes in Computer Science 1517, pages 1–16. Springer-Verlag, 1998.

[CO98] B. Courcelle and S. Olariu. Upper bounds to the clique-width of graphs. Technical report, University of Bordeaux, 1998.

[CR97] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In Ph.G. Kolaitis and F. Afrati, editors, *Database Theory - ICDT '97*, volume 1186 of *Lecture Notes in Computer Science*, pages 56–70. Springer-Verlag, 1997.

[CR98] C. Chekuri and A. Ramajaran. Conjunctive query containment revisited. Technical report, Stanford University, November 1998.

[Dec90] R. Dechter. Decomposing a relation into a tree of binary relations. *Journal of Computer and System Sciences*, 41:1–24, 1990.

[Dec92] R. Dechter. Constraint networks. In S.C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 276–185. Wiley, New York, 1992.

[DF99] R.G. Downey and M.R. Fellows. *Parametrized Complexity*. Springer-Verlag, 1999.

[DG84] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[DP89] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989.

[DP92] R. Dechter and J. Pearl. Structure identification in relational data. *Artificial Intelligence*, 48:237–270, 1992.

[Fre90] E.C Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proc. AAAI-90*, pages 4–9, 1990.

[FV93] T.A. Feder and M.Y. Vardi. Monotone monadic SNP and constraint satisfaction. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 612–622, 1993.

[FV99] T.A. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM J. on Computing*, 28:57–104, 1999.

[Gai82] H. Gaifman. On local and nonlocal properties. In J. Stern, editor, *Logic Colloquium '81*, pages 105–135. North Holland, 1982.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co., 1979.

[GJC94] M. Gyssens, P.G. Jeavons, and D.A. Cohen. Decomposition constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66:57–89, 1994.

[GLS98] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. In *Proc. 39th IEEE Symp. on Foundation of Computer Science*, pages 706–715, 1998.

[GLS99] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. In *Proc. 18th ACM Symp. on Principles of Database Systems*, pages 21–32, 1999.

[HN90] P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48:92–110, 1990.

[JC95] P. Jeavons and D. Cohen. An algebraic characterization of tractable constraints. In D.-Z. Du and M. Li, editors, *Proceedings of First Annual International Conference on Computing and Combinatorics COCOON '95*, pages 633–642. Springer-Verlag, 1995.

[JCG95] P. Jeavons, D. Cohen, and M. Gyssens. A unifying framework for tractable constraints. In U. Montanari and F. Rossi, editors, *Proceedings of 1st International Conference on Principles and Practice of Constraint Programming, CP95*, pages 276–291. Springer-Verlag, 1995.

[JCG96] P. Jeavons, D. Cohen, and M. Gyssens. A test for tractability. In E.C. Freuder, editor, *Proceedings of 2nd International Conference on Principles and Practice of Constraint Programming, CP96*, pages 267–281. Springer-Verlag, 1996.

[Jea97] P. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 1997. To appear.

[Kum92] V. Kumar. Algorithms for constraint-satisfaction problems. *AI Magazine*, 13:32–44, 1992.

[KV92] Ph. G. Kolaitis and M. Y. Vardi. Infinitary logic and 0-1 laws. *Information and Computation*, 98:258–294, 1992. Special issue: Selections from the Fifth Annual IEEE Symposium on Logic in Computer Science.

[KV95] Ph. G. Kolaitis and M. Y. Vardi. On the expressive power of Datalog: tools and a case study. *Journal of Computer and System Sciences*, 51(1):110–134, August 1995. Special Issue: Selections from Ninth Annual ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Nashville, TN, USA, 2-4 April 1990.

[KV96] Ph.G. Kolaitis and M.Y. Vardi. On the expressive power of variable-confined logics. In *Proceedings of 11th Annual IEEE Symposium on Logic in Computer Science - LICS '96*, pages 348–59, 1996.

[KW98] R.W. Kaye and R. Wilson. *Linear Algebra.* Oxford Univ. Pr., 1998.

[LMSS95] A.Y. Levy, A.O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. 14th ACM Symp. on Principles of Database Systems*, pages 95–104, 1995.

[LP97] H.R. Lewis and C. Papadimitriou. *Elements of the Theory of Computation.* Prentice Hall, 1997.

[McK43] J. McKinsey. The decision problem for some classes of sentence without quantifiers. *Journal of Symbolic Logic*, 8:61–76, 1943.

[Mes89] P. Meseguer. Constraint satisfaction problems: an overview. *AICOM*, 2:3–16, 1989.

[MF93] A.K. Mackworth and E.C. Freuder. The complexity of constraint satisfaction revisited. *Artificial Intelligence*, 59(1-2):57–62, 1993.

[Mon74] U. Montanari. Networks of constraints: fundamental properties and application to picture processing. *Information Science*, 7:95–132, 1974.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

[PJ97] J. Pearson and P. Jeavons. A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, Royal Holloway University of London, 1997.

[Qia96] X. Qian. Query folding. In *Proc. 12th Int'l Conf. on Data Engineering*, pages 48–55, 1996.

[Riv74] I. Rival. Maximal sublattices of finite distributive lattices. *Proc. AMS*, 44:263–268, 1974.

[RSU95] A. Rajamaran, Y. Sagiv, and J.D. Ullman. Answering queries using templates with binding patterns. In *Proc. 14th ACM Symp. on Principles of Database Systems*, pages 105–112, 1995.

[Sar91] Y. Saraiya. *Subtree elimination algorithms in deductive databases.* PhD thesis, Department of Computer Science, Stanford University, 1991.

[Sch78] T.J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th ACM Symp. on Theory of Computing*, pages 216–226, 1978.

[Tsa93] E.P.K. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[Ull89] J. D. Ullman. *Database and Knowledge-Base Systems, Volumes I and II.* Computer Science Press, 1989.

[Ull97] J.D. Ullman. Information integration using logical views. In Ph.G. Kolaitis and F. Afrati, editors, *Database Theory - ICDT '97*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1997.

[Var95] M.Y. Vardi. On the complexity of bounded-variable queries. In *Proc. 14th ACM Symp. on Principles of Database Systems*, pages 266–76, 1995.

[vL90] J. van Leeuwen. Graph algorithms. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science A: Algorithms and Complexity*, chapter 10, pages 525–631. Elsevier, Amsterdam, 1990.

[Yan81] M. Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal of Computing*, 10:310–327, 1981.