Constraint Satisfaction and Database Theory: a Tutorial

Moshe Y. Vardi*
Department of Computer Science
Rice University
Houston, TX 77005-1892, U.S.A.
vardi@cs.rice.edu
http://www.cs.rice.edu/~vardi/

Abstract

A large class of problems in AI and other areas of computer science can be viewed as constraint-satisfaction problems. This includes problems in machine vision, belief maintenance, scheduling, temporal reasoning, type reconstruction, graph theory, and satisfiability. In general, the constraint satisfaction-problem is NP-complete, so searching for tractable cases is an active research area. It turns out that constraint satisfaction has an intimate connection with database theory: constraint-satisfaction problems can be recast as database problems and database problems can be recast as constraint-satisfaction problems. In this tutorial, I will cover the fundamentals of constraints saisfaction and describe its intimate relationship with database theory from various perspectives.

1 Introduction

Since the early 1970s, 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 (cf. [16, 41, 46, 51]). 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 (cf. [44, 16, 18, 26, 47]).

Over the last few years, it has become clear that there is an intimate connection between constraint satisfaction and various problems in database theory. The goal of this tutorial is to describe several such connections. We start (Sections 2 and 3) by describing the constraint-satisfaction framework and show how it can be viewed from a database perspective. We then show three applications of database theory to constraint satisfaction (Sections 4, 5, and 6). We conclude with an application of constraint satisfaction to database theory (Section 7).

This tutorial is not meant as a comprehensive overview, but rather as a personal perspective. Material for this paper has been drawn from [39, 40, 10].

*Partially supported by NSF grant CCR-9700061.

2 Preliminaries

The standard terminology in AI formalizes an instance $\mathcal{P}$ of CSP as a triple $(V, D, \mathcal{C})$, consisting of a set V of variables, a set D of values, and a collection $\mathcal{C}$ of *constraints* $C_1, \ldots, C_q$, where each C_i is a pair (t, R) with t a tuple over V and R is a relation on D of the same arity as $|t|$. A *solution* of such an instance, is a mapping $h : D \rightarrow V$ such that for each constraint (t, R) in $\mathcal{C}$ we have that $h(t) \in R$, where h is defined on tuples component-wise. The constraint-satisfaction problem asks whether a given instance is *solvable*, i.e., has a solution. Note that, without loss of generality, we may assume that all constraints (t, R_i) involving a tuple t have been consolidated to a single constraint (t, R), where R is the intersection of all the constraining relations R_i. Thus, we can assume that each tuple t of variables occurs at most once in the collection $\mathcal{C}$.

An important observation made in [21] is that every such CSP instance $\mathcal{P}$ can be viewed as an instance of the *homomorphism*[1] *problem*, where we ask whether there is a homomorphism between two structures $\mathbf{A}_{\mathcal{P}}$ and $\mathbf{B}_{\mathcal{P}}$, where the domain of $\mathbf{A}_{\mathcal{P}}$ is V, the domain of $\mathbf{B}_{\mathcal{P}}$ is D, the relations of $\mathbf{B}_{\mathcal{P}}$ are the distinct relations R occurring in $\mathcal{C}$, and the relations of $\mathbf{A}_{\mathcal{P}}$ are defined as follows: for each relation R on D occurring in $\mathcal{C}$, we have the relation $R^{\mathbf{A}} = \{t : (t, R) \text{ is a constraint}\}$. We call $(\mathbf{A}_{\mathcal{P}}, \mathbf{B}_{\mathcal{P}})$ the *homomorphism instance* of $\mathcal{P}$. It is also clear that every instance of the homomorphism problem between two structures $\mathbf{A}$ and $\mathbf{B}$ can be viewed as a CSP instance CSP($\mathbf{A}, \mathbf{B}$) by simply "breaking up" each relation $R^{\mathbf{A}}$ on $\mathbf{A}$ as follows: we generate a constraint $(t, R^{\mathbf{B}})$ for each $t \in R^{\mathbf{A}}$. We call CSP($\mathbf{A}, \mathbf{B}$) the *CSP instance* of $(\mathbf{A}, \mathbf{B})$. We use both formalisms in this paper, as each has its own advantages.

[1]A homomorphism from a relation structure $\mathbf{A}$ to a relational structure $\mathbf{B}$ over the same vocabulary is a mapping from the domain of $\mathbf{A}$ to the domain of $\mathbf{B}$ such that every tuple in a relation of $\mathbf{A}$ is mapped to a tuple in the corresponding relation of $\mathbf{B}$.

It turns out that in both formulations constraint satisfaction can be expressed as a database-theoretic problem. We start with the traditional AI formulation. Suppose we are given an instance $(V, D, \mathcal{C})$. We can assume without loss of generality that in every constraint $(t, R) \in \mathcal{C}$ the elements in t are distinct. (Suppose to the contrary that $t_i = t_j$. Then we can delete from R every tuple in which the ith and jth entries disagree, and then project out that jth column from t and R.) We can thus view every element of V as a relational *attribute*, every tuple of distinct elements of V as a relational *scheme*, and every contraint (t, R) as a relation R over the scheme t (cf. [2]). It now follows from the definition of CSP that it can be viewed as a *join-evaluation problem.*

Proposition 2.1: [4, 32] *A CSP instance* $(V, D, \mathcal{C})$ *is solvable iff* $\bowtie_{(t,R)\in\mathcal{C}} R$ *is nonempty.*

On the other hand, the homomorphism formulation is intimately related to *conjunctive-query evaluation.* A 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 positive 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 relation on D obtained by evaluating the query Q on D, that is, the collection of all tuples from D that satisfy the query. Let Q_1 and Q_2 be two 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). 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 $P(X_1, Z_1, Z_2)$, $R(Z_2, Z_3)$, $R(Z_3, X_2)$, $P_1(X_1)$, $P_2(X_2)$. The relationship between conjunctive-query containment, conjunctive-query evaluation, and homomorphisms is provided by the following classical result.

Proposition 2.2: [13] *Let* Q_1 *and* Q_2 *be two conjunctive queries having the same tuple* $(X_1, \ldots, X_n)$ *of distinguished variables. Then the following statements are equivalent.*

- $Q_1 \subseteq Q_2$.
- $(X_1, \ldots, X_n) \in Q_2(D^{Q_1})$.
- *There is a homomorphism* $h : D^{Q_2} \rightarrow D^{Q_1}$.

It follows that the homomorphism problem can be viewed as a conjunctive-query evaluation problem or as a conjunctive-query containment problem. Given a pair $\mathbf{A}, \mathbf{B}$ of structure over the same vocabulty, we can view the domain $A = \{x_1, \ldots, x_n\}$ as a set of individual variables and associate with $\mathbf{A}$ the Boolean conjunctive query

$$\varphi_{\mathbf{A}} = \exists x_1 \ldots \exists x_n \bigwedge_{t \in R \text{ in } \mathbf{A}} R(t).$$

We can associate an analogous query $\varphi_{\mathbf{B}}$ with $\mathbf{B}$.

Proposition 2.3: [39] *There is a homomorphism from* $\mathbf{A}$ *to* $\mathbf{B}$ *iff* $\varphi_{\mathbf{A}}$ *is true in* $\mathbf{B}$ *iff* $\varphi_{\mathbf{B}} \subseteq \varphi_{\mathbf{A}}$.

3 Uniform vs Non-Uniform Constraint Satisfaction Problem

Let $\mathcal{A}$ and $\mathcal{B}$ be two classes of finite relational structures. The *(uniform) constraint-satisfaction problem* $\text{CSP}(\mathcal{A}, \mathcal{B})$ is the following decision problem: given a structure $\mathbf{A} \in \mathcal{A}$ and a structure $\mathbf{B} \in \mathcal{B}$, is there a homomorphism $h : \mathbf{A} \rightarrow \mathbf{B}$? Note that, by its very definition, each $\text{CSP}(\mathcal{A}, \mathcal{B})$ problem is in NP. We write $\text{CSP}(\mathcal{B})$ for the special uniform case $\text{CSP}(\mathcal{A}, \mathcal{B})$ in which $\mathcal{A}$ is the class of all finite relational structures over the vocabulary of $\mathcal{B}$. If $\mathcal{B}$ consists of a single structure $\mathbf{B}$, then we write $\text{CSP}(\mathcal{A}, \mathbf{B})$ instead of $\text{CSP}(\mathcal{A}, \{\mathbf{B}\})$. We refer to such problems as *non-uniform* constraint satisfaction problems, because the inputs are just structures $\mathbf{A}$ in $\mathcal{A}$. We also write $\text{CSP}(\mathbf{B})$ for the special non-uniform case $\text{CSP}(\mathcal{A}, \mathbf{B})$ in which $\mathcal{A}$ is the class of all finite relational structures over the vocabulary of $\mathbf{B}$. Note that if $\mathbf{B}$ is a *Boolean structure*, i.e., it has $\{0, 1\}$ as its domain, then $\text{CSP}(\mathbf{B})$ is a *generalized satisfiability* problem in the sense of Schaefer [50].

Over the past twenty years, researchers in computational complexity theory have studied non-uniform constraint satisfaction problems in an attempt to determine for which structures $\mathbf{B}$ the associated $\text{CSP}(\mathbf{B})$ problem is tractable and for which it is intractable. The first remarkable success on this front was obtained by Schaefer [50], who pinpointed the computational complexity of Boolean $\text{CSP}(\mathbf{B})$ problems. Schaefer established a *dichotomy* theorem for Boolean $\text{CSP}(\mathbf{B})$ problems. Specifically, he identified six classes of Boolean structures and showed that $\text{CSP}(\mathbf{B})$ is solvable in polynomial time, if $\mathbf{B}$ is in one of these classes, but $\text{CSP}(\mathbf{B})$ is NP-complete in all other cases. In particular, Schaefer's 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 [33] established a dichotomy theorem for $\text{CSP}(\mathbf{B})$ problems in which $\mathbf{B}$ is an *undirected* graph: if $\mathbf{B}$ is 2-colorable, then $\text{CSP}(\mathbf{B})$ is solvable in polynomial time; otherwise, $\text{CSP}(\mathbf{B})$ is NP-complete. Observe that if K_k is a clique with k nodes, then $\text{CSP}(K_k)$ 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 [21] raised two questions: (1) Under what conditions is the non-uniform porblem $\text{CSP}(\mathbf{B})$ tractable, and (2) is every

CSP($\mathbf{B}$) problem either solvable in polynomial time or NP-complete? Although they did not settle these questions, Feder and Vardi were able to isolate two conditions that imply polynomial-time solvability of CSP($\mathbf{B}$) problems; moreover, they argued that all known polynomially solvable CSP($\mathbf{B}$) problems seem to satisfy one of these conditions. The first condition is group-theoretic and covers Schaefer's tractable class of *affine* satisfiability problems. The second condition, which we dicuss at length in the next two sections, asserts that the complement of the CSP($\mathbf{B}$) problem at hand is expressible in Datalog (CSP($\mathbf{B}$) 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. (See [34, 35, 36] for another line of attack on the classification question for CSP($\mathbf{B}$) problems.)

4 Constraint Satisfaction, Games, and Datalog

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 [52]). 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). Thus, expressibility in Datalog is a sufficient condition for tractability of a query.

If $\mathbf{B}$ is a finite relational structure and $\mathcal{A}$ is a class of structures, then we write $\neg$CSP($\mathcal{A}, \mathbf{B}$) for the *complement* of CSP($\mathcal{A}, \mathbf{B}$), that is, the class of structures $\mathbf{A}$ such that there is no homomorphism $h : \mathbf{A} \to \mathbf{B}$. Feder and Vardi [21] provided a unifying explanation for the tractability of many non-uniform CSP($\mathbf{B}$) problems by showing that the complement of each of these problems is expressible in Datalog. Kolaitis and Vardi [39] then showed how Datalog also provides a unifying explanation for the uniform tractability of constraint-satisfaction problems.

Note that, in general, non-uniform tractability results do not uniformize. Thus, tractability results for each problem in a collection of non-uniform CSP($\mathbf{B}$) 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 $\mathbf{A}$ and $\mathbf{B}$ are part of the input to the constraint satisfaction problem, and the running times of the polynomial-time algorithms for CSP($\mathbf{B}$) may very well be exponential in the size of $\mathbf{B}$.

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 [12, 2]). 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 [37]). Here 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.

Theorem 4.1: [39] *Let k be a positive integer. Every k-*Datalog *query over finite structures is expressible in $\exists\mathcal{L}^k_{\infty\omega}$. Thus, k-*Datalog $\subseteq \exists\mathcal{L}^k_{\infty\omega}$.

Next, we describe certain combinatorial games that will play an important role in the sequel. Let $\mathbf{A}$ and $\mathbf{B}$ be two relational structures over a common relational vocabulary σ. The *existential k-pebble game on* $\mathbf{A}$ *and* $\mathbf{B}$ is played between two players, the *Spoiler* and the *Duplicator*. The Spoiler places k pebbles (one at a time) on elements of $\mathbf{A}$; after each move of the Spoiler, the Duplicator responds by placing a pebble on an element of $\mathbf{B}$. 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 $\mathbf{A}$ and $\mathbf{B}$ 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 $\mathbf{A}$ with domain $\{a_1, \ldots, a_k\}$ to the substructure of $\mathbf{B}$ with domain $\{b_1, \ldots, b_k\}$.

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* $\mathbf{A}$ *and* $\mathbf{B}$ if he has a strategy that allows him to continue playing "forever", that is, the Spoiler can never win a round of the game. This is made formal by the following definition.

Definition 4.2: Let k be a positive integer and let $\mathbf{A}$ and $\mathbf{B}$ be two relational structures over the same vocabulary with domains A and B respectively.

- A *winning strategy for the Duplicator in the existential k-pebble game on* $\mathbf{A}$ *and* $\mathbf{B}$ is a nonempty family of *k-partial* homomorphisms (i.e., the domain of each homomorphism has at most k elements) from A to B and the family has the *k-forth property*, which means that for every $f \in \mathcal{F}$ with $|f| < k$ and every $a \in A$ on which f is undefined, there is a $g \in \mathcal{F}$ that extends f and is defined on a.

- A *configuration for the existential k-pebble game on* $\mathbf{A}$ *and* $\mathbf{B}$ is a $2k$-tuple $\overline{a}, \overline{b}$, where $\overline{a}$ and $\overline{b}$ are elements of A^k and B^k respectively such that if $a_i = a_j$, then $b_i = b_j$ (i.e., the correspondence $a_i \mapsto b_i$, $1 \leq i \leq k$, is a partial function from A to B, which we denote by $h_{\overline{a},\overline{b}}$).
- A *winning configuration for the Duplicator in the existential k-pebble game on* $\mathbf{A}$ *and* $\mathbf{B}$ is a configuration $\overline{a}, \overline{b}$ for this game such that $h_{\overline{a},\overline{b}}$ is a member of some winning strategy for the Duplicator in this game. We denote by $\mathcal{W}^k(\mathbf{A}, \mathbf{B})$ the set of all such configurations.

∎

The following results shows that expressibility in $\exists\mathcal{L}^k_{\infty\omega}$ can be characterized in terms of the existential k-pebble games.

Proposition 4.3: [38] *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 $\mathbf{A}$, $\mathbf{B}$ are two structures in $\mathcal{C}$, $(\overline{a}, \overline{b}) \in \mathcal{W}^k(\mathbf{A}, \mathbf{B})$, and $\mathbf{A} \models Q(\overline{a})$, then $\mathbf{B} \models Q(\overline{b})$.*

Corollary 4.4: *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 $\mathbf{A}$ and $\mathbf{B}$ are two structures in $\mathcal{C}$ such that $\mathbf{A} \models Q$ and the Duplicator wins the existential k-pebble game on $\mathbf{A}$ and $\mathbf{B}$, then $\mathbf{B} \models Q$.*

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 $(\mathbf{A}, \mathbf{B})$ of two σ-structures $\mathbf{A}$ and $\mathbf{B}$ by a single $\sigma_1 + \sigma_2$-structure $\mathbf{A} + \mathbf{B}$ defined as follows: the domain of $\mathbf{A} + \mathbf{B}$ is the union of the domains of $\mathbf{A}$ and $\mathbf{B}$, the interpretation of D_1 (respectively, D_2) is the domain of $\mathbf{A}$ (respectively, $\mathbf{B}$), and the interpretation of each relation symbol R_1 (respectively, R_2) is the interpretation of the relation symbol R on $\mathbf{A}$ (respectively, on $\mathbf{B}$). This encoding makes it possible to formally view queries on pairs of σ-structures as queries on single $\sigma_1 + \sigma_2$-structures.

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

Theorem 4.5: [39] *Let σ be a relational vocabulary and let k be a positive integer.*

1. *There is a positive first-order formula $\varphi(\overline{x}, \overline{y}, S)$, where $\overline{x}$ and $\overline{y}$ are k-tuples of variables, over the vocabulary $\sigma_1 + \sigma_2$ such that the complement of its least fixpoint on a pair $\mathbf{A} + \mathbf{B}$ of structures defines the set $\mathcal{W}^k(\mathbf{A}, \mathbf{B})$.*
2. *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 $\mathbf{A}$ and $\mathbf{B}$, does the Spoiler win the existential k-pebble on $\mathbf{A}$ and $\mathbf{B}$?". As a result, there is a polynomial-time algorithm such that, given two finite σ-structures $\mathbf{A}$ and $\mathbf{B}$, it determines whether the Spoiler wins the existential k-pebble game on $\mathbf{A}$ and $\mathbf{B}$.*
3. *For every finite σ-structure $\mathbf{B}$, there is a k-*Datalog *program $\rho_{\mathbf{B}}$ that expresses the query "Given a σ-structure $\mathbf{A}$, does the Spoiler win the existential k-pebble game on $\mathbf{A}$ and $\mathbf{B}$?".*

The next theorem establishes a connection between expressibility of $\neg\mathrm{CSP}(\mathcal{A}, \mathbf{B})$ in k-Datalog and existential k-pebble games. (A closely related, but somewhat less precise, such connection was established in [21]).

Theorem 4.6: [39] *Let k be a positive integer, $\mathbf{B}$ a finite relational structure, and $\mathcal{A}$ a class of finite relational structures such that $\mathbf{B} \in \mathcal{A}$. Then the following statements are equivalent.*

1. *$\neg\mathrm{CSP}(\mathcal{A}, \mathbf{B})$ is expressible in k-*Datalog *on $\mathcal{A}$.*
2. *$\neg\mathrm{CSP}(\mathcal{A}, \mathbf{B})$ is expressible in $\exists\mathcal{L}^k_{\infty\omega}$ on $\mathcal{A}$.*
3. *$\neg\mathrm{CSP}(\mathcal{A}, \mathbf{B}) = \{\mathbf{B} \in \mathcal{A}$: The Spoiler wins the existential k-pebble game on $\mathbf{A}$ and $\mathbf{B}\}$.*

By combining Theorems 4.5 and 4.6, we obtain the following uniform tractability result for classes of constraint satisfaction problems expressible in Datalog.

Theorem 4.7: [39] *Let k be a positive integer, $\mathcal{A}$ a class of finite relational structures, and*

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

Then the uniform constraint satisfaction problem $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$ *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 $\mathbf{A}$ and $\mathbf{B}$.*

We note that it is an open problem whether the class $\{\mathbf{B} : \neg\mathrm{CSP}(\mathcal{A}, \mathbf{B})$ is expressible in $k - \mathrm{Datalog}\}$ is recursive.

In the next section, we take a closer look at the connection between constraint satisfaction, games, and Datalog.

5 Datalog and Consistency

One of the most fruitful approaches to coping with the intractability of constraint satisfaction has been the introduction and use of various *consistency* concepts that make explicit additional constraints implied by the original constraints. The connection between consistency properties and tractability was first described in [23, 24]. In a similar vein, the relationship between *local consistency* and *global consistency* is investigated in [17, 54, 55]. Intuitively, local consistency means that any partial solution on a set of variables can be extended to a partial solution containing an additional variable, whereas global consistency means that any partial solution can be extended to a global solution. Note that if the inputs are such that local consistency implies global consistency, then there is a polynomial-time algorithm for constraint satisfaction; moreover, in this case a solution can be constructed via a backtrack-free search. We now describe this approach from the Datalog perspective.

The crucial insight is that the key concept of *strong k-consistency* [17] is equivalent to a property of winning strategies for the Duplicator in the existential k-pebble game. Specifically, an instance of a constraint-satisfaction problem is strongly k-consistent if and only if the family of *all* k-partial homomorphims f is a winning strategy for the Duplicator in the existential k-pebble game on the two relational structures that represent the given instance. The connection between pebble games and consistency properties, however, is deeper than just a mere reformulation of the concept of strong k-consistency. Indeed, as mentioned earlier, consistency properties underly the process of making explicit new constraints that are implied be the original constraints. A key technical step in this approach is the procedure known as "establishing strong k-consistency", which propagates the original constraints, adds implied constraints, and transforms a given instance of a constraint satisfaction problem to a stronly k-consistent instance with the same solution space [15, 17]. In fact, strong k-consistency can be established if and only if the Duplicator wins the existential k-pebble game. Moreover, whenever strong k-consistency can be established, one method for doing this is to first compute the largest winning strategy for the Duplicator in the existential k-pebble game and then modify the original problem by augmenting it with the constraints expressed by the largest winning strategy; this method gives rise to the least constrained instance that establishes strong k-consistency and, in addition, satisfies a natural *coherence* property. By combining this result with known results concerning the definability of the largest winning strategy, it follows that the algorithm for establishing strong k-consistency in this way (with k fixed) is actually expressible in least fixed-point logic; this strengthens the fact that strong k-consistency can be established in polynomial time, when k is fixed. If we consider non-uniform constraint satisfaction, it follows that for every relational structure $\mathbf{B}$, the complement of CSP($\mathbf{B}$) is expressible by a Datalog program with k variables if and only if CSP($\mathbf{B}$) coincides with the collection of all relational structures $\mathbf{A}$ such that establishing strong k-consistency on $\mathbf{A}$ and $\mathbf{B}$ implies that there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$.

We start the formal treatment with the following observation.

Proposition 5.1: [40] *If $\mathcal{F}$ and $\mathcal{F}'$ are two winning strategies for the Duplicator in the existential k-pebble game on two structures $\mathbf{A}$ and $\mathbf{B}$, then also the union $\mathcal{F} \cup \mathcal{F}'$ is a winning strategy for the Duplicator. Hence, there is a largest winning strategy for the Duplicator in the existential k-pebble game, namely the union of all winning strategies, which is precisely $\mathcal{H}^k(\mathbf{A},\mathbf{B}) = \{h_{\overline{a},\overline{b}} : (\overline{a},\overline{b}) \in \mathcal{W}^k(\mathbf{A},\mathbf{B})\}$.*

We now recall the concepts of *i-consistency* and *strong k-consistency.*

Definition 5.2: Let $\mathcal{P} = (V, D, \mathcal{C})$ be a CSP instance. $\mathcal{P}$ is *i-consistent* if for every $i-1$ variables $v_1, \ldots, v_{i-1}$, for every partial solution on these variables, and for every variable $v_i \notin \{v_1, \ldots, v_{i-1}\}$, there is a partial solution on the variables $v_1, \ldots, v_{i-1}, v_i$ extending the given partial solution on the variables $v_1, \ldots, v_{i-1}$. $\mathcal{P}$ is *strongly k-consistent* if it is i-consistent for every $i \leq k$. ∎

A key insight is that these concepts can be naturally recast in terms of existential pebble games.

Proposition 5.3: [40] *Let $\mathcal{P}$ be a CSP instance, and let $(\mathbf{A}_{\mathcal{P}}, \mathbf{B}_{\mathcal{P}})$ be the associated homomorphism instance. $\mathcal{P}$ is i-consistent if and only if the family of all partial homomorphisms from $\mathbf{A}_{\mathcal{P}}$ to $\mathbf{B}_{\mathcal{P}}$ with $i-1$ elements in their domain has the i-forth property. $\mathcal{P}$ is strongly k-consistent if and only if the family of all k-partial homomorphisms from $\mathbf{A}_{\mathcal{P}}$ to $\mathbf{B}_{\mathcal{P}}$ is a winning strategy for the Duplicator in the existential k-pebble game on $\mathbf{A}_{\mathcal{P}}$ and $\mathbf{B}_{\mathcal{P}}$.*

Let us now recall the concept of *establishing strong k-consistency*, as defined, for instance, in [15, 17]. This concept has been defined rather informally in the literature to mean that, given an instance $\mathcal{P}$ of CSP, we associate an instance $\mathcal{P}'$ that has the following properties: (1) $\mathcal{P}'$ has the same set of variables and the same set of values as $\mathcal{P}$ (2) $\mathcal{P}'$ is strongly k-consistent; (3) $\mathcal{P}'$ is more constrained than $\mathcal{P}$; and (4) $\mathcal{P}$ and $\mathcal{P}'$ have the same space of solutions. The next definition formalizes the above concept in the context of the homomorphism problem.

Definition 5.4: Let $\mathbf{A}$ and $\mathbf{B}$ be two relational structures over a k-ary vocabulary σ (i.e., every relation symbol in σ has arity at most k). *Establishing strong k-consistency for* $\mathbf{A}$ *and* $\mathbf{B}$ means that we associate two relational structures $\mathbf{A}'$ and $\mathbf{B}'$ with the following properties:

1. $\mathbf{A}'$ and $\mathbf{B}'$ are structures over some k-ary vocabulary σ' (in general, different than σ); moreover, the domain of $\mathbf{A}'$ is the domain A of $\mathbf{A}$, and the domain of $\mathbf{B}'$ is the domain B of $\mathbf{B}$.
2. CSP($\mathbf{A}', \mathbf{B}'$) is strongly k-consistent.
3. if h is a k-partial homomorphism from $\mathbf{A}'$ to $\mathbf{B}'$, then h is a k-partial homomorphism from $\mathbf{A}$ to $\mathbf{B}$.
4. If h is a function from A to B, then h is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ if and only if h is a homomorphism from $\mathbf{A}'$ to $\mathbf{B}'$.

If the structures $\mathbf{A}'$ and $\mathbf{B}'$ have the above properties, then we say that $\mathbf{A}'$ *and* $\mathbf{B}'$ *establish strong k-consistency for* $\mathbf{A}$ *and* $\mathbf{B}$. ∎

An instance $\mathcal{P}$ of CSP is *coherent* if every constraint (t, R) of $\mathcal{P}$ completely determines all constraints (u, Q) in which all variables occurring in u are among the variables of t. We formalize this concept as follows.

Definition 5.5: An instance $\mathbf{A}, \mathbf{B}$ of the homomorphism problem is *coherent* if its associated CSP instance CSP($\mathbf{A}, \mathbf{B}$) has the following property: for every constraint $(\overline{a}, R)$ of CSP($\mathbf{A}, \mathbf{B}$) and every tuple $\overline{b} \in R$, the mapping $h_{\overline{a},\overline{b}}$ is well defined and is a partial homomorphism from $\mathbf{A}$ to $\mathbf{B}$. ∎

Note that a CSP instance can be made coherent by polynomial-time constraint propagation.

The main result of this section is that strong k-consistency can be established precisely when the Duplicator wins the existential k-pebble game. Moreover, one method for establishing strong k-consistency is to first compute the largest winning strategy for the Duplicator in this game and then generate an instance of the constraint-satisfaction problem consisting of all the constraints embodied in the largest winning strategy. Furthermore, this method gives rise to the largest coherent instance that establishes strong k-consistency (and, hence, the least constrained such instance).

Theorem 5.6: [40] *Let k be a positive integer, let σ be a k-ary vocabulary, and let $\mathbf{A}$ and $\mathbf{B}$ be two relational structures over σ with domains A and B, respectively. It is possible to establish strong k-consistency for $\mathbf{A}$ and $\mathbf{B}$ if and only if $\mathcal{W}^k(\mathbf{A},\mathbf{B}) \neq \emptyset$. Furthermore, if $\mathcal{W}^k(\mathbf{A},\mathbf{B}) \neq \emptyset$, then the following sequence of steps gives rise to two structures $\mathbf{A}'$ and $\mathbf{B}'$ that establish strong k-consistency for $\mathbf{A}$ and $\mathbf{B}$:*

1. *Compute the set $\mathcal{W}^k(\mathbf{A},\mathbf{B})$.*
2. *For every $i \leq k$ and for every i-tuple $\overline{a} \in A^i$, form the set $R_{\overline{a}} = \{\overline{b} \in B^i : (\overline{a},\overline{b}) \in \mathcal{W}^k(\mathbf{A},\mathbf{B})\}$.*

3. *Form the CSP instance $\mathcal{P}$ with A as the set of variables, B as the set of values, and $\{(\overline{a}, R_{\overline{a}}) : \overline{a} \in \cup_{i=1}^{k} A^i\}$ as the collection of constraints.*

4. *Let $(\mathbf{A}', \mathbf{B}')$ be the homomorphism instance of $\mathcal{P}$.*

In addition, the structures $\mathbf{A}'$ and $\mathbf{B}'$ obtained above constitute the largest coherent instance establishing strong k-consistency for $\mathbf{A}$ and $\mathbf{B}$, that is, if $(\mathbf{A}'', \mathbf{B}'')$ is another such coherent instance, then for every constraint $(\overline{a}, R)$ of $\mathrm{CSP}(\mathbf{A}'', \mathbf{B}'')$*, we have that $R \subseteq R_{\overline{a}}$.*

The key step in the procedure described in Theorem 5.6 is the first step, in which the set $\mathcal{W}^k(\mathbf{A}, \mathbf{B})$ is computed. The other steps simply "re-format" $\mathcal{W}^k(\mathbf{A}, \mathbf{B})$. From Theorem 4.5 it follows that we can establish strong k-consistency by computing the fixed-point of a monotone first-order formula. We can now relate the concept of strong k-consistency to the results in [21] regarding Datalog and non-uniform CSP.

Theorem 5.7: [40] *Let $\mathbf{B}$ be a relational structure over a vocabulary σ. $\neg\mathrm{CSP}(\mathbf{B})$ is expressible in k-Datalog iff for every structure $\mathbf{A}$ over σ, establishing strong k-consistency for $\mathbf{A}, \mathbf{B}$ implies that there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$.*

6 Bounded Treewidth and Constraint Satisfaction

Up to this point, we found tractable cases of the uniform constraint satisfaction problem $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$ by imposing restrictions on the class $\mathcal{B}$. In this section, we exhibit tractable cases of $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$ 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 [21] and generalizes the concept of treewidth of a graph (see [56, 5]).

A *tree decomposition* of a finite relational structure $\mathbf{A}$ 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 domain 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.

For every $k \geq 1$, let $\mathcal{A}(k)$ be the class of all finite relational structures of treewidth k. Bodlaender [5] 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.

It has been shown in [25] that a uniform constraint-satisfaction problem $\mathrm{CSP}(\mathcal{A}(k), \mathcal{B})$ is tractable, independetly of $\mathcal{B}$ (see also [19]). Here we describe a proof of this result via a connection with first-order queries with a bounded number of distinct variables.

Let $\mathbf{A}$ and $\mathbf{B}$ be two finite relational structure. Recall that by Proposition 2.3, the existence of a homomorphism $h : A \rightarrow B$ is equivalent to whether $\varphi_{\mathbf{A}}(\mathbf{B})$ is true, where $\varphi_{\mathbf{A}}$ is the Boolean conjunctive query whose body consist of the conjunction of all facts in $\mathbf{A}$. Consider the fragment $\exists\mathrm{FO}_{\wedge,+}$ of first-order logic, which allows no negative formulas, no disjunctions, and no universal quantifiers. It is easy to see that fragment has the same expressive power as conjunctive queries. It turns out that the fragment $\exists\mathrm{FO}^{k+1}_{\wedge,+}$ of $\exists\mathrm{FO}_{\wedge,+}$, where we allow at most $k+1$ individual variables in formulas, can express precisely the queries $\varphi_{\mathbf{A}}$, where $\mathbf{A}$ is a structure of treewidth k.

Proposition 6.1: [25] *Let $\mathbf{A}$ be a finite relational structure. Then $\mathbf{A}$ has treewidth k iff $\varphi_{\mathbf{A}}$ is expressible in $\exists\mathrm{FO}^{k+1}_{\wedge,+}$.*

We can now derive the main result of this section.

Theorem 6.2: [39] *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 asserted in Proposition 6.1 that if $\mathbf{A}$ is a finite relational structure of treewidth k, then $\varphi_{\mathbf{A}}$ is equivalent to an $\exists\mathrm{FO}^{k+1}_{\wedge,+}$ formula. In fact, the formulas can be constructed efficiently from *parse trees*, which can be constructed efficiently from tree decompositions of structures of bounded treewdith [39]. Thus, a $\exists\mathrm{FO}^{k+1}_{\wedge,+}$ formula equivalent to $\varphi_{\mathbf{A}}$ can be constructed in time polynomial in the size of $\mathbf{A}$. Thus, in this case, checking the existence of a homomorphism $h : \mathbf{A} \rightarrow \mathbf{B}$ reduces to the evaluation of a $\exists\mathrm{FO}^{k+1}_{\wedge,+}$ query on the structure $\mathbf{B}$. As shown in [58], $\exists\mathrm{FO}^{k+1}$ 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 $\mathrm{CSP}(\mathcal{A}(k), \mathcal{F})$ is provided in [28], 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). Note that, in contrast, the combined complexity of evaluating queries in FO^k (the k-variable fragment of first-order logic) is PTIME-complete [58].

The study of the impact of the "topology" of a conjunctive queries on the complexity of their evaluation dates to the study of the complexity of evaluating *acyclic* joins [45]. The connection between acyclic joins and acyclic constraints was pointed out in [32]. This is still an active research area. Chekuri and Ramajaran [14] showed that the uniform constraint satisfaction problem $\mathrm{CSP}(\mathcal{Q}(k), \mathcal{F})$ is solvable in polynomial time, where $\mathcal{Q}(k)$ is the class of structures of *querywidth* k. They also showed that the incidence treewidth of a structure $\mathbf{A}$ 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 [5], while the property of having querywidth 4 is NP-complete [30].) Gottlob, Leone, and Scarcello [30] define another notion of width, called *hypertree width.* They showed that the querywidth of a structure $\mathbf{A}$ provides a strict upper bound for the hypertree width of $\mathbf{A}$, but that the class $\mathcal{H}(k)$ of structures of hypertree width at most k is polynomially recognizable, and that $\mathrm{CSP}(\mathcal{H}(k), \mathcal{F})$ is tractable. For further discussion on the relative merit of various notions of "width", see [29]. At this point, hypertree width seems to be the most powerful way to obtain tractability results for constraint satisfaction using the "topology" of the input instance.

7 Constraint Satisfaction and View-based Query Processing

Several recent papers in the literature show that the problem of view-based query processing [53, 1] is relevant in many aspects of database management, including including query optimization, data warehousing, data integration, and query answering with incomplete information. Informally speaking, the problem requires to answer a query posed to a database only on the basis of the information on a set of views, which are again queries over the same database.

There are two approaches to view-based query processing, called *query rewriting* and *query answering*, respectively. In the former approach, we are given a query Q and a set of view definitions, and the goal is to reformulate the query into an expression that refers only to the views, and provides the answer to Q. Typically, the rewriting is expressed in the same language used for both the query and the views. In the latter approach, besides Q and the view definitions, we are also given the extensions of the views. The goal is to compute the set of tuples that are implied by these extensions.

In the last years a large number of results have been reported for both problems. Query rewriting has been studied under different assumptions on the form of the queries and views, cf. [43, 48, 53]. A comprehensive framework for view-based query answering, as well as several interesting results, are presented in [1, 31]. In [8, 9, 11] view-based query processing has been studied for the case of regular-path queries (RPQs).

We consider a setting in which databases are expressed in terms of edge-labeled graphs, and queries ask for pairs of nodes connected by a specified path. This setting is typical in *semistructured data*, where all data models share the characteristic that data are organized in a labeled graph, where the nodes represent objects, and the edges represent links between objects [7]. Semistructured data models and have been introduced with the aim of capturing data on the Web, digital libraries, and the like. The main difficulty arising in this context is that languages for querying semistructured data enable expressing RPQs [3, 6, 22]. An RPQ asks for all pairs of nodes in the database connected by a path conforming to a regular expression, and therefore may contain a restricted form of recursion. Note that when the query contains unrestricted recursion, both view-based query rewriting and view-based query answering become undecidable, even when the views are not recursive [20].

In this section we study the relationship between view-based query rewriting and view-based query answering. We define a *rewriting* of a query with respect to a set of views as a function that, given the extensions of the views, returns a set of pairs of objects that is contained in the answer set of the query with respect to the views. We call the rewriting that returns exactly such set the *perfect* rewriting of the query wrt the views. Thus, view-based query answering, and evaluating the perfect rewriting over given view extensions, are equivalent problems.

Typically, one is interested in queries that are PTIME functions (in data complexity). Hence, we would like rewritings to be PTIME as well. From this the relationship between view-based query rewriting and view-based query answering, and by the results in [9], it follows that perfect rewritings are not PTIME in general. Hence, the problem arises of characterizing which instances of query rewriting admit a perfect rewriting that is PTIME. Unfortuanetly, this question seems to be quite hard, as it is intimately related to the attempt to characterize tractable non-uniform constraint-satisfaction problems.

Formally, we consider a *database* as an edge labeled graph $DB = (\mathcal{D}, \mathcal{E})$, where $\mathcal{D}$ is a set of nodes (called the *domain*) that represent the objects of DB, and $\mathcal{E} = \{r_e \mid e \in \Sigma\}$ is a set of binary relations corresponding to the edges of the graph labeled by elements from an alphabet Σ. Such edges represent links between objects labeled by attribute names. We denote an edge from node x to node y labeled by r, i.e., $(x, y) \in r$, with $x \xrightarrow{r} y$.

As query mechanism we consider *regular-path queries* (RPQs), which are the basic constituents of full-fledged query languages over semistructured data [7]. Such queries denote all the paths corresponding to words of a specified regular language over the alphabet Σ, and hence are expressed by means of regular expressions or finite automata. The *answer set of an RPQ Q over a database DB* is $ans(Q, DB) = \{(x, y) \mid$ there is a path $x \xrightarrow{r_1} \cdots \xrightarrow{r_n} y$ in DB s.t. $r_1 \cdots r_n \in L(Q)\}$, where $L(Q)$ is the regular language defined by Q.

Next we introduce the problem of view-based query answering [1, 31, 42]. Consider a database that is accessible only through a set $\mathcal{V} = \{V_1, \ldots, V_k\}$ of views, and suppose we want to answer an RPQ only on the basis of our knowledge on the views. Specifically, associated to each view V_i we have:

- its definition $def(V_i)$ in terms of an RPQ over the alphabet Σ;
- information about its extension in terms of a set $ext(V_i)$ of pairs of objects[2].

We use $def(\mathcal{V})$ to denote $(def(V_1), \ldots, def(V_k))$, $ext(\mathcal{V})$ to denote $(ext(V_1), \ldots, ext(V_k))$, and $\mathcal{D}_{\mathcal{V}}$ to denote the set of objects appearing in $ext(\mathcal{V})$.

We say that a database DB is *consistent with the views* $\mathcal{V}$ if $ext(V_i) \subseteq ans(def(V_i), DB)$, for each $V_i \in \mathcal{V}$. The *certain answer set of Q wrt the views $\mathcal{V}$* is the set $cert(Q, \mathcal{V}) \subseteq \mathcal{D}_{\mathcal{V}} \times \mathcal{D}_{\mathcal{V}}$ such that $(c, d) \in cert(Q, \mathcal{V})$ if and only if $(c, d) \in ans(Q, DB)$, for every DB that is consistent with $\mathcal{V}$.

The problem of *view-based query answering* is the following: Given

- a set $\mathcal{V}$ of views, their definitions $def(\mathcal{V})$, and extensions $ext(\mathcal{V})$,
- a query Q,
- a pair of objects $c, d \in \mathcal{D}_{\mathcal{V}}$,

decide whether $(c, d) \in cert(Q, \mathcal{V})$.

The complexity of the problem can be measured in three different ways [57]:

- *Data complexity*: as a function of the size of $ext(\mathcal{V})$.
- *Expression complexity*: as a function of the size of Q and of the expressions in $def(\mathcal{V})$.
- *Combined complexity*: as a function of the size of $ext(\mathcal{V})$, Q, and $def(\mathcal{V})$.

In [9] the following complexity characterization of view-based query answering is given.

Theorem 7.1: [9] *View-based query answering for RPQs is co-NP-complete in data complexity and PSPACE-complete in expression and combined complexity.*

[2] We assume that objects are represented by constants, and we adopt the *unique name assumption* [49], i.e., different constants denote different objects and therefore different nodes.

The definition of view-based query answering given above reflects two implicit assumptions: (i) The views are *sound*, i.e., from the fact that a pair (a, b) is in $ext(V_i)$ we can conclude that (a, b) is in $ans(def(V_i), DB)$, but not vice-versa. (ii) The domain is *open*, i.e., a database consistent with the views may contain additional objects that do not appear in the view extensions. Other assumptions about the accurateness of the knowledge on the objects of the database and the pairs satisfying the views, have been studied [1, 31, 9].

We now describe the relationship between view-based query answering and query rewriting. An instance of *query rewriting* is given by a query Q and a set $\mathcal{V}$ of views with definitions $def(\mathcal{V})$. One then tries to generate a new query Q' over the symbols in $\mathcal{V}$ such that Q' approximates the answer to Q, when V_i is interpreted as $ext(V_i)$, for each $V_i \in \mathcal{V}$. Formally, we require $ans(Q', ext(\mathcal{V})) \subseteq cert(Q, \mathcal{V})$. In the context of RPQs, Q and $def(V_1), \ldots, def(V_k)$ are regular expressions over the alphabet Σ, while Q' is a regular expression over the alphabet $\mathcal{V}$.

A solution to the problem of *RPQ rewriting* is described in [8], where an algorithm is given to compute the maximal RPQ rewriting wrt to all rewritings that are RPQs. Nevertheless, such rewriting is in general not maximal if we allow for rewritings that belong to a larger class of queries. From a more abstract point of view, we can define a *rewriting of Q wrt $\mathcal{V}$* as a function that, given $ext(\mathcal{V})$, returns a set of pairs of objects that is contained in the certain answer set $cert(Q, \mathcal{V})$. We call the rewriting that returns exactly $cert(Q, \mathcal{V})$ the *perfect rewriting* of Q wrt $\mathcal{V}$. The problem of *view-based query rewriting* is the one of computing a rewriting of Q wrt $\mathcal{V}$. The problem comes in different forms, depending of the properties that we require for the rewriting. In particular:

- It is sometimes interesting to consider rewritings that are expressible in a certain query language, e.g., Datalog.
- It is also interesting to consider rewritings belonging to a certain data complexity class, for example, polynomial time. A rewriting f belongs to a data complexity class $\mathcal{C}$ if the problem of deciding whether a pair of objects (c, d) is in $f(ext(\mathcal{V}))$ is in the class $\mathcal{C}$, where the complexity of the problem is measured with respect to the size of $ext(\mathcal{V})$.
- Finally, it is worth computing rewritings that are maximal in a certain class. A rewriting f of Q wrt $\mathcal{V}$ is *maximal in a class* $\mathcal{C}$ if, for every rewriting $g \in \mathcal{C}$ of Q wrt $\mathcal{V}$, we have that $g(ext(\mathcal{V})) \subseteq f(ext(\mathcal{V}))$ for every $ext(\mathcal{V})$.

An algorithm for view-based query answering is an algorithm that takes as input a query, a set of view definitions, and a set of view extensions, and determines whether a given pair of objects is in the answer set of the query for every database that is consistent with the views. Hence, if we fix the query and the view definitions, we can consider every algorithm for view-based query answering as an algorithm that computes whether a given pair of objects is in the perfect rewriting. This observation establishes a tight connection between view-based query answering and query rewriting.

Now, considering that in the present setting view-based query answering is co-NP-complete in data complexity (see Theorem 7.1), we obtain the following result.

Theorem 7.2: [9] *The perfect rewriting of an RPQ wrt RPQ views is a co-NP function. There is an RPQ Q and a set $\mathcal{V}$ of RPQ views such that the rewriting of Q wrt $\mathcal{V}$ is a co-NP-complete function.*

Typically, one is interested in queries that are PTIME functions. Hence, we would like rewritings to be PTIME as well. Unfortunately, by Theorem 7.2, perfect rewritings are not PTIME in general (assuming PTIME≠NPTIME). Hence it would be interesting to characterize which instances of query rewriting admit a perfect rewriting that is PTIME. Note, however, that finding such instances corresponds to finding those instances of view-based query answering that are PTIME in data complexity. We now show that this is going to be difficult, by exhibiting a tight connection between view-based query answering and constraint satisfaction.

We show first that every CSP is polynomially reducible to view-based query answering.

Theorem 7.3: [10] *Let $\mathbf{B}$ be a directed graph. There exists an RPQ Q and RPQ views $\mathcal{V}$ with definitions $def(\mathcal{V})$ such that the following holds: for every directed graph $\mathbf{A}$, there are extensions $ext(\mathcal{V})$ and objects c, d such that $(c, d) \notin cert(Q, \mathcal{V})$ if and only if CSP$(\mathbf{A}, \mathbf{B})$ is solvable.*

The reduction in the proof of Theorem 7.3 is polynomial, so we get the following corollary.

Corollary 7.4: [10] *Every CSP over directed graphs is polynomially reducible to view-based query answering.*

We note that it is shown in [21] that constraint-satisfaction problems over directed graphs are just as hard as general constraint-satisfaction problems.

We show next a reduction from view-based query answering to CSP. To this end, given a query Q and a set $\mathcal{V}$ of views with definitions $def(\mathcal{V})$, we call the *constraint template* of Q wrt $\mathcal{V}$ the structure $\mathbf{B}$ defined as follows. The vocabulary of $\mathbf{B}$ is $\mathcal{V} \cup \{U_c, U_d\}$, where symbols in $\mathcal{V}$ denote binary predicates, and U_c and U_d denote unary predicates. Let $A_Q = (\Sigma, S, S_0, \rho, F)$ be a (nondeterministic) automaton for Q. The structure $\mathbf{B} = (B, \mathcal{V}^{\mathbf{B}})$ is given by:

- The domain B of $\mathbf{B}$ is 2^S;
- $(\sigma_1, \sigma_2) \in V_i{}^{\mathbf{B}}$ iff there exists a word $w \in L(def(V_i))$ such that $\rho(\sigma_1, w) \subseteq \sigma_2$;
- $\sigma \in U_c{}^{\mathbf{B}}$ iff $S_0 \subseteq \sigma$, and $\sigma \in U_d{}^{\mathbf{B}}$ iff $\sigma \cap F = \emptyset$.

Theorem 7.5: [10] *Let Q be an RPQ and $\mathcal{V}$ a set of RPQ views with definitions $def(\mathcal{V})$. Then the problem of verifying, given $ext(\mathcal{V})$ and objects c, d, whether $(c, d) \notin cert(Q, \mathcal{V})$ is polynomially reducible to CSP$(\mathbf{B})$, where $\mathbf{B}$ is the constraint template of Q wrt $\mathcal{V}$.*

Corollary 7.6: [10] *View-based query answering is polynomially reducible to CSP.*

Theorems 7.3 and 7.5 exhibit a very strong connection between CSP and view-based query answering. In particular, since in the reduction in Theorem 7.3, the query and the view definitions depend only on graph $\mathbf{B}$, and only the view extensions depend on graph $\mathbf{A}$, the theorem shows that non-uniform CSP can be polynomially reduced to query rewriting. As a consequence, if we had a method to decide whether an instance of query rewriting admits a perfect rewriting that is PTIME, we would then be able to precisely characterize those instances of non-uniform CSP that are in PTIME. As discussed above, this is a longstanding open problem that appears to be difficult to solve. On the other hand, however, it is shown in [10] how the connection between CSP and Datalog described in Section 4 can be used to derive (non-perfect) Datalog rewritings for RPQs with respect to RPQ views.

References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM Sym. on Principles of Database Systems (PODS'98)*, pages 254–265, 1998.

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

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

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

[5] 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.

[6] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization technique for unstructured data. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 505–516, 1996.

[7] P. Buneman, D. Suciu, and S. Abiteboul. *Data on the Web : From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[8] D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of the 18th ACM Sym. on Principles of Database Systems (PODS'99)*, pages 194–204, 1999.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. Answering regular-path queries using views. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 389–398, 2000.

[10] D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. View-based query processing and constraint satisfaction. Submitted, 2000.

[11] D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. View-based query processing for regular-path queries with inverse. In *Proc. of the 19th ACM Sym. on Principles of Database Systems (PODS 2000)*, 2000. This proceedings.

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

[13] 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.

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

[15] M.C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41(1):89–95, 1989.

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

[17] R. Dechter. From local to global consistency. *Artificial Intelligence*, 55(1):87–107, May 1992.

[18] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211–241, 1994.

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

[20] O.M. Duschka and M.R. Genesereth. Answering recursive queries using views. In *Proc. of the 16th ACM Sym. on Principles of Database Systems (PODS'97)*, pages 109–116, 1997.

[21] T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM J. on Computing*, 28:57–104, 1999. Preliminary version in *Proc. 25th ACM Symp. on Theory of Computing*, May 1993, pp. 612–622.

[22] M.F. Fernandez, D. Florescu, J. Kang, A.Y. Levy, and D. Suciu. Catching the boat with strudel: Experiences with a web-site management system. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 414–425, 1998.

[23] E.C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21(11):958–966, November 1978.

[24] E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery*, 29(1):24–32, 1982.

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

[26] D.H. Frost. *Algorithms and Heuristics for Constraint Satisfaction Problems*. PhD thesis, Department of Computer Science, University of California, Irvine, 1997.

[27] M.L. Ginsberg, editor. *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann, Los Altos, 1987.

[28] 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.

[29] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. In *Proc. 16th Int'l Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 394–399, 1999.

[30] 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.

[31] G. Grahne and A.O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 332–347. Springer-Verlag, 1999.

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

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

[34] 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.

[35] 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.

[36] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–48, 1997.

[37] 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.

[38] 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.

[39] Ph.G. Kolaitis and M.Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proc. 17th ACM Symp. on Principles of Database Systems*, pages 205–13, 1998. Full version at `http://www.cs.rice.edu/~vardi/papers`.

[40] Ph.G. Kolaitis and M.Y. Vardi. A game-theoretic approach to constraint satisfaction. Submitted, 2000.

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

[42] A.Y. Levy. Obtaining complete answers from incomplete databases. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, pages 402–412, 1996.

[43] 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.

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

[45] D. Maier. *The Theory of Relational Databases.* Computer Science Press, Rockville, Md., 1983.

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

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

[48] A. Rajaraman, Y. Sagiv, and J.D. Ullman. Answering queries using templates with binding patterns. In *Proc. of the 14th ACM Sym. on Principles of Database Systems (PODS'95)*, 1995.

[49] R. Reiter. On closed world data bases. In Hervé Gallaire and Jack Minker, editors, *Logic and Databases*, pages 119–140. Plenum Publ. Co., New York, 1978. Republished in [27].

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

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

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

[53] J.D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1997.

[54] P. van Beek. On the inherent tightness of local consistency in constraint networks. In *Proc. of National Conference on Artificial Intelligence (AAAI-94)*, pages 368–373, 1994.

[55] P. van Beek and R. Dechter. Constraint tightness and looseness versus local and global consistency. *Journal of the ACM*, 44(4):549–566, 1997.

[56] 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.

[57] M.Y. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM Sym. on Theory of Computing (STOC'82)*, pages 137–146, 1982.

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