UNIVERSITY OF CALIFORNIA
SANTA CRUZ

COMPLEXITY IN DATABASES, GAMES, AND LOGICS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Jonathan T. Panttaja

September 2006

The Dissertation of Jonathan T. Panttaja
is approved:

Professor Phokion G. Kolaitis, Chair

Professor Wang-Chiew Tan

Professor Luca De Alfaro

Lisa C. Sloan
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Complexity in Databases, Games, and Logics

by

Jonathan T. Panttaja

Databases are ubiquitous in our society. This dissertation presents a complexity analysis of two important problems related to databases. First, we show the complexity of establishing strong k-consistency, which is a common heuristic for database query evaluation. Second, we classify the complexity of the existence-of-solutions problem for data exchange.

The first part of this dissertation presents a computational complexity analysis of the problem of determining the winner of the existential k-pebble game. Existential k-pebbles were originally introduced to aid in the analysis of the expressive power of Datalog and related infinitary logics with finitely many variables. It has since been observed that strong k-consistency can be established for an instance of constraint satisfaction if and only if the Duplicator has a winning strategy for the existential k-pebble game on structures related to the given instance of constraint satisfaction. The main result is that determining the winner of the existential k-pebble game is EXPTIME-complete when k is part of the input. This implies that establishing strong k-consistency is inherently exponential. We then use this result to prove complexity results for a collection of related games.

The second part of this dissertation addresses the combined complexity of the existence-of-solutions problem for data exchange. Data exchange is the problem of transforming data from a source schema to a target schema so that all constraints of a schema mapping are satisfied. The existence-of-solutions problem is the question: given a source instance, is there a target instance that satisfies the constraints of the schema mapping? Earlier work showed that for schema mappings specified by embedded implicational dependencies, this problem is solvable in polynomial time, if the schema mapping is fixed and the constraints of the schema mapping satisfy a certain structural condition, called weak acyclicity. We show that if the schema mapping is also part of the input, then the complexity of the existence-of-solutions problem rises to 2EXPTIME. We also show that there are restricted classes of inputs to the existence-of-solutions problem for which the problem can be EXPTIME-complete or CoNP-complete.

Acknowledgments

The first part of this dissertation is based on joint work with Phokion Kolaitis, published in [19]. The second part is based on joint work with Phokion Kolaitis and Wang-Chiew Tan, published in [20]. This work was supported in part by NSF grants IIS-0430994 and IIS-9907419.

Part I

Constraint Satisfaction and Games

Chapter 1

Introduction

When retrieving data from a relational database, we write a conjunctive query that specifies the data we want, and then evaluate that query against the database. The problem of deciding whether the answer to a conjunctive query is non-empty is NP-complete in general when both the query and the database are given as input. This query evaluation problem turns out to be equivalent to another problem, known as the Constraint Satisfaction Problem. The Constraint Satisfaction Problem (CSP) has been deeply studied in the area of artificial intelligence. The idea is that you are given variables and constraints on the values those variables may take. The question is whether there is an assignment of values to variables that satisfies all the constraints. Because it is equivalent to the query evaluation problem, CSP is also NP-complete in general. On the other hand, we know that there are subproblems that are solvable efficiently. For instance, if no constraint involves more than one variable, we can easily decide if there is a solution.

There are, in fact, many large classes of subproblems for which an efficient algorithm exists to solve CSP, but because the problem is hard in general, researchers have come up with heuristics to solve problems. One of the most effective classes of heuristics involves consistency properties. To talk about consistency, we must first consider the concept of a partial solution. A partial solution is an assignment of values to some of the variables in a CSP instance. The size of a partial solution is the number of variables assigned values. A CSP instance is called k-consistent, if every partial solution of size $k-1$ can be extended to a partial solution of size k. An instance is called strongly k-consistent if every partial solution of size less than k can be extended to a partial solution of size k. An instance is called globally consistent if

any partial solution can be extended to a solution to the instance. Dechter, in [5], showed that there are conditions where strong k-consistency implies global consistency. This gives rise to interest in algorithms to take an instance of CSP and try to change it into an equivalent instance that is strongly k-consistent. Cooper [4] has presented an algorithm that establishes strong k-consistency in exponential time. Unfortunately, Cooper's algorithm is not efficient enough to be useful on general problems. It would be useful to know if we can do better.

While studying Datalog, Kolaitis and Vardi [23] found a connection to help with this issue. They introduced the existential k-pebble game, to help study Datalog and related logics. This is a game played on two relational structures $\mathbf{A}$ and $\mathbf{B}$ by two players known as Spoiler and Duplicator. They take turns placing pebbles on the structures, Spoiler playing on $\mathbf{A}$ and Duplicator playing on $\mathbf{B}$. Duplicator wins by always making sure that the partial mapping from $\mathbf{A}$ to $\mathbf{B}$ determined by the pebbles is always a partial homomorphism. As it turns out, determining strong k-consistency for an instance of CSP is equivalent to determining a winning strategy for the Duplicator in a corresponding instance of the existential k-pebble game.

In [19], Kolaitis and I showed that determining the winner of the existential k-pebble game is EXPTIME-complete when k is part of the input, and P-complete when k is fixed. This shows that unless the complexity hierarchy collapses, we cannot improve asymptotically on Cooper's algorithm for establishing strong k-consistency. There is another, similar, game known as the k-pebble game. Grohe [15] showed that determining the winner of this game is P-complete when k is fixed, but the complexity when k is part of the input is still unknown.

1.1 Constraint Satisfaction

An *instance of the Constraint Satisfaction Problem* consists of a set V of variables, a set C of constraints, and a domain D of possible values for the variables. Each constraint is a pair (v, R) where $R \subseteq D^k$ is a k-ary relation over D, and $v \in V^k$ is a k tuple of variables from V. This represents the fact that the set of variables in v must take on the values from one of the tuples of R. The question we want to answer is: Given an instance of CSP, is there an assignment of values to variables such that every constraint is satisfied? More precisely, is there a mapping $h : V \to D$, such that for every constraint (v, R), $h(v) \in R$?

Example 1.1.1 *Consider the CSP instance with* $V = \{v_1, v_2, v_3\}$, $D = \{0, 1\}$, *and*

$$\begin{array}{r|r|r} C_1 = (v_1, v_2), R_1 & C_2 = (v_2, v_3), R_2 & C_3 = (v_1, v_3), R_3 \\ R_1 = \{(0,0), & R_2 = \{(1,0), & R_3 = \{(1,0), \\ (0,1), & (0,1), & (0,1)\} \\ (1,0)\} & (1,1)\} & \end{array}$$

There are two solutions: $v_1 = 0, v_2 = 0, v_3 = 1$, *and* $v_1 = 0, v_2 = 1, v_3 = 1$.

For much of what follows we will be considering problems on relational structures. Relational structures are a generalization of directed graphs. A *relational structure* is a tuple $\mathbf{A} = (A, R_1, \ldots, R_n)$, such that A is a set of domain elements, and each $R_i \subseteq A^{k_i}$. The *vocabulary* of a relational structure is the set of names of relations in the structure, along with their arities.

Feder and Vardi [12] showed that CSP is equivalent to the Homomorphism Problem. The Homomorphism Problem is the problem of determining whether, given two relational structures $\mathbf{A}$ and $\mathbf{B}$, there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$. For this problem, a *homomorphism* is a mapping h from A to B such that for every tuple $(a_1, \ldots, a_k) \in R_i^A$, we have that $(h(a_1), \ldots, h(a_k)) \in R_i^B$.

Example 1.1.2 *Consider the structure* $\mathbf{A} = (A, R^A)$, *where* A *is* $\{a_1, a_2, a_3\}$ *and the relation* R^A *is* $\{(a_1, a_2), (a_2, a_1), (a_2, a_3), (a_3, a_2)\}$. *Consider also the strucutre* $\mathbf{B} = (B, R^B)$, *such that the domain* B *is* $\{b_1, b_2, b_3\}$, *and the relation* R^B *is* $\{(b_2, b_3), (b_3, b_2), (b_1, b_3), (b_3, b_1)\}$.

Let h *be the mapping from* $\mathbf{A}$ *to* $\mathbf{B}$ *such that* $h(a_i) = b_i$. *This is not a homomorphism, because* $(a_1, a_2) \in R^A$, *but* $(h(a_1), h(a_2)) = (b_1, b_2) \notin R^B$.

The mapping h' *that maps* a_1 *to* b_1, a_2 *to* b_3, *and* a_3 *to* b_2, *is a homomorphism.*

This instance of the homomorphism problem does have a homomorphism.

Example 1.1.3 *Let* $\mathbf{A} = (A, R^A)$ *be a structure with domain* A *equal to* $\{a_1, a_2, a_3\}$ *a relation* R^A *that contains all possible distinct pairs of elements of* A. *Let* $\mathbf{B} = (B, R^B)$ *be a structure such that the domain* B *is* $\{b_1, b_2, b_3\}$, *and* R^B *equals* $\{(b_1, b_2), (b_2, b_1), (b_2, b_3), (b_3, b_2)\}$.

This instance of the homomorphism problem does not have a solution. Note that there is an edge between each pair of distinct elements of A, *and there are no self loops in* $\mathbf{B}$. *This means that any mapping that maps two elements of* A *to the same element of* B *cannot be a homomorphism. Thus, any mapping of elements of* A *to elements of* B *would map some* a_i *to* b_3, *and some* a_j *to* b_1. *Since by definition, there is a tuple* $(a_i, a_j) \in R^A$, *but* $(b_3, b_1) \notin R^B$, *we know that no mapping can be a homomorphism.*

Example 1.1.4 *We can also represent k-colorability as a homomorphism problem. Let $\mathbf{B} = K_k$, where K_k is the complete graph on k vertices. So, for any graph, $\mathbf{A}$ the question of whether there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ is equivalent to the question of whether $\mathbf{A}$ is k colorable.*

If $k = 2$ this is 2-colorability, which is in polynomial time.

If $k = 3$ this is 3-colorability, which is NP-complete.

In the transformation from CSP to homomorphism, $\mathbf{A}$ corresponds to the variables of the instance of CSP, and to the tuples of variables in the constraints. In $\mathbf{B}$, we have the values that the variables may take on, along with the relations from the constraints. A homomorphism from $\mathbf{A}$ to $\mathbf{B}$ corresponds to a mapping of variables to values that satisfies all the constraints.

More specifically, given an instance of CSP with variables V, domain D, and constraints $C = \{(\mathbf{x_1}, R_1), \ldots (\mathbf{x_n}, R_n)\}$, we construct an instance of the homomorphism problem as follows. We construct the relational structures $\mathbf{A} = (V, \{\mathbf{x_1}\}, \ldots, \{\mathbf{x_n}\})$ and $\mathbf{B} = (D, R_1, \ldots, R_n)$. This is a simple restructuring of the data. Any assignment of values to variables can be considered a mapping from V to D. If we consider this a mapping h from $\mathbf{A}$ to $\mathbf{B}$, then h is a homomorphism if and only if the original assignment is a satisfying assignment. To see this, consider a tuple of variables $\mathbf{x_i} = (x_1, \ldots, x_k)$. For h to be a homomorphism, $(h(x_1), \ldots, h(x_k)) \in R_i$. But this is exactly the criterion for the assignment to be satisfying.

If we wish to consider the relationship in the other direction, we simply consider a pair of relational structures $\mathbf{A} = (A, R_1^A, \ldots, R_n^A)$, and $\mathbf{B} = (B, R_1^B, \ldots, R_n^B)$, and transform them into an instance of CSP. We let A be the set of variables and B be the set of values. We then construct $C = \{(\mathbf{x_{ij}}, R_i^B) | \mathbf{x_{ij}} \in R_i^A\}$; that is, we build a constraint for each tuple in each relation from $\mathbf{A}$.

It is easy to show that CSP is in the class NP. We must show that given an assignment, we can decide whether it is a satisfying assignment in polynomial time. For any constraint $((x_1, \ldots, x_k), R)$ and assignment of values to variables h, we simply check to see if the tuple $(h(x_1), \ldots, h(x_k)) \in R$. This can be accomplished by a simple linear search for each constraint.

CSP contains, as a special case, 3-SAT. Given a 3-CNF formula $\phi(\mathbf{x})$, the variables of the CSP instance are just the variables of the formula, the domain is simply $\{0, 1\}$, and the relations correspond to ternary or, with all possible combinations of positive and negative literals. Table 1.1 shows four relations. There are, of course, eight possible clauses in a CNF

Clause	$R_1 = (x \vee y \vee z)$	$R_2 = (\neg x \vee y \vee z)$	$R_3 = (\neg x \vee \neg y \vee z)$	$R_4 = (\neg x \vee \neg y \vee \neg z)$
Relation	$(1,1,1)$	$(1,1,1)$	$(1,1,1)$	$(1,1,0)$
	$(1,1,0)$	$(1,1,0)$	$(1,0,1)$	$(1,0,1)$
	$(1,0,1)$	$(1,0,1)$	$(1,0,0)$	$(1,0,0)$
	$(1,0,0)$	$(0,1,1)$	$(0,1,1)$	$(0,1,1)$
	$(0,1,1)$	$(0,1,0)$	$(0,1,0)$	$(0,1,0)$
	$(0,1,0)$	$(0,0,1)$	$(0,0,1)$	$(0,0,1)$
	$(0,0,1)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$

Table 1.1: Converting SAT clauses into CSP constraints

formula, but because $\vee$ is commutative, we can rearrange each clause, based on the number of negations, into one of these four.

Example 1.1.5 *Given $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (x_4 \vee \neg x_2 \neg x_3)$, we construct an instance of CSP as follows. The set of variables $V = \{x_1, x_2, x_3, x_4\}$, the domain $D = \{0, 1\}$, and the constraints $C = \{((x_1, x_2, x_3), R_1), ((x_1, x_3, x_4), R_2), ((x_2, x_3, x_4), R_3)\}$.*

Because 3-SAT is NP-complete, this reduction shows us that CSP is NP-complete in general.

One common approach in the study of NP-complete problems is to consider restrictions of the problem. If we think about CSP as the homomorphism problem, there are two inputs, $\mathbf{A}$ and $\mathbf{B}$. This is uniform CSP. We could instead consider the non-uniform problem where one of the inputs is fixed. The number of possible mappings from $\mathbf{A}$ to $\mathbf{B}$ is $|B|^{|A|}$. So, if we fix $\mathbf{A}$, the homomorphism problem is solvable in polynomial time in the size of $\mathbf{B}$.

If, instead we consider non-uniform CSP with $\mathbf{B}$ fixed, also known as CSP($\mathbf{B}$), we get a more varied complexity theory. If $\mathbf{B}$ is the complete graph on two vertices, then CSP($\mathbf{B}$) corresponds to the question of whether $\mathbf{A}$ is 2-colorable. This problem is known to be solvable in polynomial time. On the other hand, as we saw earlier in the reduction from 3-SAT to CSP, there is a fixed $\mathbf{B}$, with 4 relations, for which CSP($\mathbf{B}$) is NP-complete. In fact, Thomas Schaefer [26] showed that for any $\mathbf{B}$ with a Boolean (or 2 element) domain, CSP($\mathbf{B}$) is either solvable in polynomial time or is NP-complete. Feder and Vardi considered this and introduced what is known as the Feder-Vardi Dichotomy Conjecture for non-uniform CSP.

Conjecture 1.1.1 *[11] For every* $\mathbf{B}$*, either CSP*$(\mathbf{B})$ *is solvable in polynomial time, or CSP*$(\mathbf{B})$ *is NP-complete.*

Bulatov [2] has shown that this conjecture is true for all $\mathbf{B}$ with up to a three-element domain. For higher order domains, a complete classification is still not known, but there have been efforts to find broad classes of tractable cases.

Many tractable cases of CSP($\mathbf{B}$) can be described using closure properties of the constraints. Feder and Vardi [11] described a number of tractable cases of CSP($\mathbf{B}$), many of which were later described by Jeavons et al. [16] using closure properties. These include the case of constraints closed under affine operations, semi-lattices, and near-unanimity functions.

Several of these tractable cases are subsumed by expressibility in Datalog, which we define in the next section. For sets of constraints $\mathbf{B}$ closed under near-unanimity functions or semi-lattice functions, the complement of CSP($\mathbf{B}$) is expressible in Datalog [11]. On the other hand, for sets of constraints $\mathbf{B}$ closed under affine functions, the complement of CSP($\mathbf{B}$) is not necessarily expressible in Datalog [11].

1.2 Datalog

A *Datalog program* consists of a finite number of *rules*. Each rule has the form $R_0(\mathbf{x}) :- R_1(\mathbf{x_1}) \ldots R_n(\mathbf{x_n})$. For each rule, we refer to $R_0(\mathbf{x})$ as the *head* of the rule, and $R_1(\mathbf{x_1}) \ldots R_n(\mathbf{x_n})$ as the *body*. A *fact* is a rule with an empty body. Each $R_i(\mathbf{x_i})$ is known as an *atom*. The elements of each $\mathbf{x_i}$ are either variables or constants. An atom that has only constants is called a *ground atom*. A fact that contains only a ground atom is known as a *ground fact*.

The relational symbols that appear in atoms of a Datalog rule are given names based on where they appear. Any relational symbol that appears in the head of a rule is called an *intentional database predicate (IDB)*, while a relational symbol that only ever appears in the body of rules is called an *extensional database predicate (EDB)*. The rules of the Datalog program tell the relationship between the EDBs and the IDBs. Given relations for the EDBS, the Datalog program tells us how to construct the IDBs.

In each Datalog program, one of the IDBs is designated as the goal. The Datalog program then defines a query over the EDBs. This query can be computed in polynomial time

in the size of the EDBs, using a simple bottom up evaluation. For each rule, we check to see if there are tuples in the relations in the body that make the body true. We then add the appropriate tuple to the relation represented in the head of the rule. One step of scanning to see if there is a rule that can be applied takes time polynomial in the size of the given EDBs. Because the arities of the relations are fixed, the number of possible tuples is a polynomial in the size of the EDBs. This tells us that the result can be computed in polynomial time.

Example 1.2.1 *Consider a Datalog program with the following three rules:*

$Q(x, y) :- E(x, y)$

$Q(x, y) :- Q(x, w), E(w, z), E(z, y)$

$G :- Q(x, x)$

This program has one EDB, E, and two IDBs, Q and G. The IDB G is the goal. Given an edge relation of a graph, this Datalog program determines if there is an odd cycle.

This is the negation of the question of whether a graph is 2-colorable. As we saw earlier the 2-colorability problem is equivalent to CSP(K_2).

Example 1.2.2 *Consider the Path Systems problem defined by Cook [3]:*

Given a finite set of formulas F, a set of axioms $A \subseteq F$, and a rule of inference $R \subseteq F^3$, compute the theorems of this system.

We can define a Datalog program to solve this as follows:

$T(x) :- A(x)$

$T(x) :- A(y), A(z), R(x, y, z)$

Cook showed that this problem is P-complete.

So we now see that finding solutions to Datalog programs is possible in polynomial time, and also there are Datalog programs that are P-complete. This gives us expressibility of a problem in Datalog as a sufficient condition for tractability. Unfortunately, CSP($\mathbf{B}$) can not be represented in Datalog. This is due to the fact that queries expressible in Datalog have the monotone property. This means that adding tuples to the EDBs cannot reduce the number of tuples in the IDBs. On the other hand, in CSP($\mathbf{B}$), adding tuples to the input instance, $\mathbf{A}$ can make a satisfiable instance unsatisfiable.

Example 1.2.3 *Let us reconsider the problem of 2-colorability. This corresponds to CSP($\mathbf{B}$) where $\mathbf{B}$ is the complete graph on two vertices. Let $\mathbf{A}$ be a path on three vertices. Every path is*

2-colorable, so there is a homomorphism from $\mathbf{A}$ *to* $\mathbf{B}$. *If we then add an edge to* $\mathbf{A}$, *and make it a cycle on three vertices, then there is no homomorphism from* $\mathbf{A}$ *to* $\mathbf{B}$.

On the other hand, if we instead consider the complement of 2-colorability, and an input graph that is not 2-colorable, it is clear that adding edges to the graph can never make it 2-colorable.

So, we have now seen that $\neg\text{CSP}(\mathbf{B})$ can be expressed in Datalog for some $\mathbf{B}$. Now, we can ask another question: When can $\neg\text{CSP}(\mathbf{B})$ be expressed in Datalog? The answer to this question comes from Kolaitis and Vardi [24], and depends on the existential k-pebble game, which we describe in the next section.

1.3 The Existential k-Pebble Game

The existential k-pebble game, or $(\exists, k)$-pebble game, was introduced by Kolaitis and Vardi in [21] to aid in the study of the expressive power of Datalog and the infinitary logic $\exists L^{\omega}_{\infty\omega}$. The logic $\exists L^{\omega}_{\infty\omega}$ consists of all logical sentences, with finitely many variables, constructed from atomic formulae using and, or, infinitary and, infinitary or, and existential quantification. For each k, we define $\exists L^{k}_{\infty\omega}$ to be the fragment of $\exists L^{\omega}_{\infty\omega}$ with k variables. Kolaitis and Vardi [21] showed that every query expressible in Datalog is also expressible in $\exists L^{\omega}_{\infty\omega}$.

Let $\mathbf{A}$ and $\mathbf{B}$ be two relational structures over the same vocabulary. A relational structure $\mathbf{A'} = \{(A', R_1^{A'}, \ldots, R_n^{A'})\}$ is a *substructure of* $\mathbf{A} = \{A, R_1^{A}, \ldots, R_n^{A}\}$ if $A' \subseteq A$ and for each i, $R_i^{A'} \subseteq R_i^{A}$. A *partial homomorphism from* $\mathbf{A}$ *to* $\mathbf{B}$ is a homomorphism from a substructure of $\mathbf{A}$ to a substructure of $\mathbf{B}$.

Let $k \geq 2$ be a positive integer. The *existential k-pebble game* (or, in short, the $(\exists, k)$-*pebble game*) is played between two players, the Spoiler and the Duplicator, on two relational structures $\mathbf{A}$ and $\mathbf{B}$ according to the following rules:

- Each player has k pebbles labeled $1, \ldots, k$.
- On the i-th move of a round of the game, $1 \leq i \leq k$, the Spoiler places a pebble on an element a_i of A, and the Duplicator responds by placing the pebble with the same label on an element b_i of B.

- The Spoiler wins the game at the end of that round, if the correspondence $a_i \mapsto b_i$, $1 \leq i \leq k$, is not a homomorphism between the substructures of $\mathbf{A}$ and $\mathbf{B}$ with universes $\{a_1, \ldots, a_k\}$ and $\{b_1, \ldots, b_k\}$, respectively.

- Otherwise, the Spoiler removes one or more pebbles, and a new round of the game begins.

The Duplicator wins the $(\exists, k)$-pebble game if he has a *winning strategy*, that is to say, a systematic way that allows him to sustain playing "forever," so that the Spoiler can never win a round of the game.

To illustrate this game (and its asymmetric character), let $\mathbf{K}_m$ be the m-clique, that is, the complete undirected graph with m nodes. For every $k \geq 2$, the Duplicator wins the $(\exists, k)$-pebble game on $\mathbf{K}_k$ and $\mathbf{K}_{k+1}$, but the Spoiler wins the $(\exists, k+1)$-pebble game on $\mathbf{K}_{k+1}$ and $\mathbf{K}_k$. As another example, let $\mathbf{L}_s$ be the s-element linear order, $s \geq 2$. If $m < n$, then the Duplicator wins the $(\exists, 2)$-pebble game on $\mathbf{L}_m$ and $\mathbf{L}_n$, but the Spoiler wins the $(\exists, 2)$-pebble game on $\mathbf{L}_n$ and $\mathbf{L}_m$.

The $(\exists, k)$-pebble game can be used to characterize when two structures satisfy the same sentences in the logic $\exists L^k_{\infty\omega}$.

Theorem 1.3.1 *[21]*

Let $\mathbf{A}$ and $\mathbf{B}$ be two relational structures over the same vocabulary. The Duplicator wins the $(\exists, k)$-pebble game on $\mathbf{A}$ and $\mathbf{B}$ if and only if $\mathbf{B}$ satisfies all of the same sentences of $\exists L^k_{\infty\omega}$ as $\mathbf{A}$. That is, for all $\phi \in \exists L^k_{\infty\omega}$, $\mathbf{A} \models \phi \Rightarrow \mathbf{B} \models \phi$.

Note that the above description of a winning strategy for the Duplicator in the $(\exists, k)$-pebble game is rather informal. The concept of a winning strategy can be made precise, however, in terms of families of partial homomorphisms with appropriate properties. Specifically, a *winning strategy for the Duplicator in the existential k-pebble game on $\mathbf{A}$ and $\mathbf{B}$* is a nonempty family $\mathcal{F}$ of partial homomorphisms from $\mathbf{A}$ to $\mathbf{B}$ such that:

1. For every $f \in \mathcal{F}$, the domain $\mathrm{dom}(f)$ of f has at most k elements.

2. $\mathcal{F}$ is *closed under subfunctions*, which means that if $g \in \mathcal{F}$ and $f \subseteq g$, then $f \in \mathcal{F}$.

3. $\mathcal{F}$ has the *k-forth property*, which means that for every $f \in \mathcal{F}$ with $|\mathrm{dom}(f)| < k$ 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.

Intuitively, the second condition provides the Duplicator with a "good" move when the Spoiler *removes* a pebble from an element of $\mathbf{A}$, while the third condition provides the Duplicator with a "good" move when the Spoiler *places* a pebble on an element of $\mathbf{A}$.

A *configuration for the* $(\exists, k)$*-pebble game on* $\mathbf{A}$ *and* $\mathbf{B}$ is a $2k$-tuple of elements from $\mathbf{A}$ and $\mathbf{B}$, $(a_1, \ldots, a_k, b_1, \ldots, b_k)$, such that $a_i \in A$, $b_i \in B$, and the mapping from A to B that maps a_i to b_i is a partial function. A *winning configuration for the Duplicator* is a configuration of the $(\exists, k)$-pebble game such that the corresponding homomorphism is a member of some winning strategy for the Duplicator. Let $\mathcal{W}^k(\mathbf{A}, \mathbf{B})$ be the set of all winning configurations.

With the definition of the $(\exists, k)$-pebble game, we can now revisit the question of when $\neg$CSP($\mathbf{B}$) can be expressed in Datalog. First we define *k-Datalog* as the fragment of Datalog in which only k variables are allowed in each rule. Kolaitis and Vardi [24] showed that there is a connection between expressibility in k-Datalog and the $(\exists, k)$-pebble game.

Theorem 1.3.2 *[24] Let* $\mathbf{B}$ *be a relational structure over a vocabulary* σ. $\neg CSP(\mathbf{B})$ *is expressible in* k*-Datalog if and only if the following condition holds:*

For every structure $\mathbf{A}$ *over* σ *if the Duplicator wins the* $(\exists, k)$*-pebble game on* $\mathbf{A}$ *and* $\mathbf{B}$*, then there is a homomorphism from* $\mathbf{A}$ *to* $\mathbf{B}$.

1.4 Consistency

To complete the connection between CSP, Datalog and the $(\exists, k)$-pebble game, we now discuss consistency properties of CSP instances. A CSP instance is *i-consistent* [13] if every partial solution of size $i-1$ can be extended to a solution of size i. Here, a *partial solution* is an assignment of values to a subset of the variables that satisfies all of the constraints over those variables. More rigorously, for every set $\{v_1, \ldots, v_{i-1}\}$ of $i-1$ variables, and any partial solution over those variables, and for any variable $v \notin \{v_1, \ldots, v_{i-1}\}$, there is a partial solution over $\{v_1, \ldots, v_{i-1}, v\}$ that is an extension of the original partial solution.

A CSP instance is known as *strongly k-consistent* [13] if it is i-consistent for all $0 \leq i \leq k$. In an instance that is strongly k-consistent, every partial solution of size less than k can be extended to a partial solution of size k. Strong 2-consistency is also known as *arc consistency*, and strong 3-consistency is known as *path consistency*. If we have an instance

of CSP over n variables, and that instance is strongly n-consistent, then that means that every partial solution can be extended to a full solution. We refer to such an instance as *globally consistent*.

Example 1.4.1 *Let $R_1 = \{(0,0),(0,1),(1,1),(1,2)\}$, and $R_2 = \{(0,1),(2,2)\}$. Consider the CSP instance $(\{x,y\}, R_1(x,y), R_2(x,y))$. This instance is not arc consistent, because 2 is a valid assignment for y, but there is no valid assignment for x associated with $y = 2$.*

Dechter [5] presented a sufficient condition for local consistency to imply global consistency. So, if an instance of CSP is strongly k-consistent for large enough k, then the instance is also globally consistent. In an instance that is globally consistent, we can perform backtrack free search to find a solution.

This property makes it desirable to have an efficient algorithm to determine whether an instance of CSP is strongly k-consistent for arbitrary k. In addition, we might ask whether we can take an instance of CSP and construct an equivalent instance that is strongly k-consistent. By equivalent, we mean that the two instances have exactly the same set of solutions. This process is known as establishing strong k-consistency, or constraint propagation. Cooper [4] presented an algorithm for establishing strong k-consistency. This algorithm takes time polynomial in the size of the instance, and exponential in k. So, for fixed k, this yields a polynomial time algorithm and for varied k, this yields an exponential algorithm.

Example 1.4.2 *Recall the previous example: $(\{x,y\}, R_1(x,y), R_2(x,y))$. The new instance $(\{x,y\}, R_1'(x,y), R_2'(x,y))$, with $R_1' = \{(0,1)\}$ and $R_2' = \{(0,1)\}$ establishes arc consistency for the initial instance.*

Even when we cannot use strong k-consistency to establish global consistency, it can be used as a heuristic to improve search. Dechter [6] presents an algorithm called bucket elimination that encompasses many algorithms, sometimes known as adaptive consistency algorithms, that make use of constraint propagation to reduce the amount of search needed to solve an instance of CSP. Thus, there is a tradeoff between search and establishing higher levels of consistency.

To tie this back to Datalog and the $(\exists, k)$-pebble game, we look to a paper by Kolaitis and Vardi [23]. They present a crucial connection between the $(\exists, k)$-pebble game and strong k-consistency.

Theorem 1.4.1 *[23] 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. It is possible to establish strong k-consistency for $\mathbf{A}$ and $\mathbf{B}$ if and only if $W^k(\mathbf{A}, \mathbf{B}) \neq$*

This result, combined with Theorem 1.3.2, gives us the following connection between expressibility in k-Datalog and establishing strong k-consistency.

Theorem 1.4.2 *[23]*

Let $\mathbf{B}$ be a relational structure over a vocabulary σ. $\neg CSP(\mathbf{B})$ is expressible in k-Datalog if and only if for every structure A over σ, establishing strong k-consistency for $\mathbf{A}, \mathbf{B}$ implies that there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$.

Chapter 2

Complexity of Pebble Games

In this chapter we show the complexity of a variety of pebble games. First, we show that determining whether the Duplicator has a winning strategy for the $(\exists, k)$-pebble game is P-complete when k is fixed. Recall that strong 2-consistency is known as arc-consistency (AC). Kasif showed that establishing arc consistency is P-complete. In the section 2.1, we present an alternate proof of this for all fixed k. This proof illustrates the ideas needed for the primary result of this dissertation, which we prove in section 2.2: If k is part of the input, determining whether the Duplicator has a winning strategy for the $(\exists, k)$-pebble game is EXPTIME-complete. We then present a framework for pebble games and discuss the complexity of the games in this framework.

2.1 Complexity of the $(\exists, k)$-Pebble Game with k Fixed

In this section we show that, for every $k \geq 2$, determining the winner of the $(\exists, k)$-pebble game is a P-complete problem. We do this by constructing a reduction from the Monotone Circuit Value problem (MCV) in the style of Grohe [15], but with different gadgets. In this reduction, the structures will be undirected graphs with ten unary predicates, called *colors*. So, we actually prove that, for every $k \geq 2$, the $(\exists, k)$-pebble game restricted to such structures is P-complete.

As was mentioned earlier, strong 2-consistency is called arc consistency. Kasif showed in [18] that establishing arc consistency is complete for polynomial time. Specifically, he showed the following theorem.

Theorem 2.1.1 *[18] The propositional Horn clause satisfiability problem is log-space reducible to AC.*

Proof: (Sketch)

An instance of the propositional Horn clause satisfiability problem (PHSP) is similar to a Datalog program. The main difference is that atoms are simply propositional variables rather than predicates. In addition, a rule of the form $\leftarrow P_0$ is known as a goal. An atom is known as *solvable* if we can establish its truth through the application of rules. A PHSP program is known as unsatisfiable if and only if the goal is solvable.

Given an instance of PHSP, Pr, we construct an instance of AC such that Pr is unsatisfiable if and only if the goal P_0 does not have a valid assignment of f. To encode this problem as an instance of AC, we construct a variable for each atom, each fact, each goal, and the body of each rule. We then add constraints which encode the possible relative values that the variables can take. For instance, if there is a rule of the form $P \leftarrow R$, then we have a constraint $((\langle P \rangle, \langle R \rangle), \{(t, f), (t, t), (f, f)\})$.

When all of these constraints are put together, establishing arc consistency is equivalent to determining whether the goal is solvable.

□

The result presented here applies to all fixed k, and also illustrates techniques used in the next section to prove the EXPTIME-hardness result.

We now present some required definitions. The following concepts and terminology come from Grohe [15].

1. In an undirected graph with colors, a *distinguished pair* of vertices is a pair of vertices that are of the same color, and of a color not used for any other vertex in the graph.

2. A *position* of the $(\exists, k)$-pebble game on A and B is a set P of ordered pairs such that $P \subseteq A \times B$ and $|P| \leq k$. Often we will omit the ordered pair notation and use the shorthand $ab \in P$ to mean that $(a, b) \in P$.

3. A *strategy* for the Spoiler is simply a mapping from positions to moves that tells the Spoiler how to play given the current position.

4. We say that the Spoiler *can reach a position P' from another position P of the $(\exists, k)$-pebble game on A and B* if the Spoiler has a strategy for the $(\exists, k)$-pebble game on $\mathbf{A}$ and $\mathbf{B}$ such that, starting from position P, either he wins the game, or after a number of moves the game is in a position P'' such that $P' \subseteq P''$.

   This concept will be used to combine strategies of the Spoiler on different gadgets to construct strategies for the combined game.

5. We say that the Duplicator *can avoid a position P' from another position P of the $(\exists, k)$-pebble game on A and B* if the Duplicator has a winning strategy for the $(\exists, k)$-pebble game on $\mathbf{A}$ and $\mathbf{B}$ such that, starting from position P, position P' never occurs.

For each gadget used in the reduction there will be two pieces, one for the Spoiler's structure and one for the Duplicator's structure. For gadget X, we call the Spoiler's side X_S, the Duplicator's side X_D, and the pair (X_S, X_D) simply X.

2.1.1 The Gadgets H and I

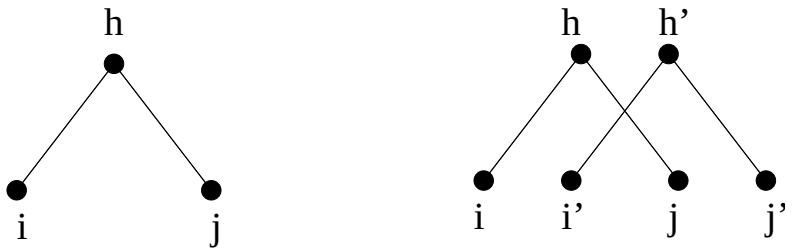

Figure 2.1: H gadget based on the gadget from [15]. H_S is on the left and H_D is on the right.

The graphs H_D and I_D, which are both based on gadgets from [15], will be used for **and** nodes and **or** nodes respectively. H_D, as seen in Figure 2.1, consists of the six vertices h, h', i, i', j, j'. These six vertices form three distinguished pairs, (h, h'), (i, i'), and (j, j'). There are edges from h to i, and h to j, and edges from h' to i' and h' to j'. This graph has only one non-identity automorphism, which we will call swi, that maps any vertex a to a' and any vertex a' to a. H_S is simply the subgraph of H_D determined by h, i, j. Starting from position hh', the Spoiler can reach both ii' and jj' in the $(\exists, k)$-pebble game on (H_S, H_D).

I_D, seen in Figure 2.2, has ten vertices. It contains the three distinguished pairs (h, h'), (i, i'), and (j, j'), plus four additional nodes, which we will name by their connections

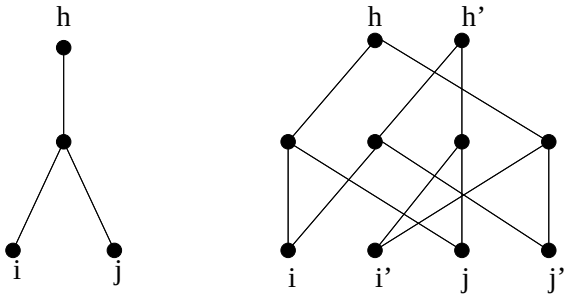

Figure 2.2: I gadget based on the gadget from [15]. I_S is on the left and I_D is on the right.

to the other vertices. These nodes are hij, $h'ij'$, $h'i'j$, and $hi'j'$. This graph has three non-identity automorphisms, which we will refer to as fix_i, fix_j, and fix_h. These automorphism fix i, j, and h respectively, while switching the other two. By playing according to these automorphisms, the Duplicator can avoid either ii' or jj' from hh', but not both, in the $(\exists, k)$-pebble game.

2.1.2 Single-Input One-Way Switches

Single-Input One-Way Switches are used to restrict the ways in which the Spoiler can win the game. The basic intuition is that the Spoiler can only make progress in one particular direction; moreover, to do so he must use all of his pebbles.

This lemma is similar to Lemma 14 from [15], adapted to the $(\exists, k)$-pebble game.

Lemma 2.1.2 *For every $k \geq 2$ there exists a pair of graphs O_S^k and O_D^k with $O_S^k \subset O_D^k$, $\{x, x', y, y'\} \subset V(O_D^k)$, xx' and yy' distinguished pairs of vertices, and $\{x, y\} \subset V(O_S^k)$, such that:*

1. *The Spoiler can reach yy' from xx' in the $(\exists, k)$-pebble game on (O_S^k, O_D^k).*
2. *There exist two disjoint sets of positions of the $(\exists, k)$-pebble game on (O_S^k, O_D^k), called Pretrapped and Trapped positions such that:*
   (a) *Pretrapped and Trapped positions are partial homomorphisms.*
   (b) *From Pretrapped positions, the Duplicator can avoid positions that are not Trapped and not Pretrapped.*

(c) From Trapped positions, the Duplicator can avoid positions that are not Trapped.

(d) The position $\{xx'\}$ is Pretrapped.

(e) If P is Pretrapped and $|P| < k$, then $P \cup \{yy\}$ is Pretrapped.

(f) The positions $\{yy\}$ and $\{yy'\}$ are Trapped.

(g) If P is Trapped and $|P| < k$, then $P \cup \{xx\}$ is Trapped.

Proof: First, we define O_S^k and O_D^k. Then, we show that these graphs have the desired properties.

$$\begin{aligned}
V(O_D^k) &= \{x, x', y, y'\} \cup \\
&\quad \{1,2\} \times \{0, 1, 2, \ldots, k-1\} \\
E(O_D^k) &= \{(x, (1,a))\} \cup \\
&\quad \{(x', (1,a)) | a > 0\} \cup \\
&\quad \{((1,a),(1,b)) | a \neq b\} \\
&\quad \{((1,a),(2,b)) | b \neq a\} \cup \\
&\quad \{((1,0),(2,0))\} \cup \\
&\quad \{((2,a), y) | a > 0\} \cup \\
&\quad \{((2,a), y'\}
\end{aligned}$$

$$\begin{aligned}
V(O_S^k) &= \{x, y\} \cup \\
&\quad \{1\} \times \{1, 2, \ldots k-1\} \cup \\
&\quad \{(2,0)\} \\
E(O_S^k) &= \{(x, (1,a))\} \cup \\
&\quad \{((1,a),(1,b)) | a \neq b\} \\
&\quad \{((1,a),(2,0))\} \cup \\
&\quad \{((2,0), y)\}
\end{aligned}$$

We color each row of vertices a unique color so that when the Spoiler places a pebble on row 2, the Duplicator cannot place a pebble on row 1. These graphs are in Figure 2.3.

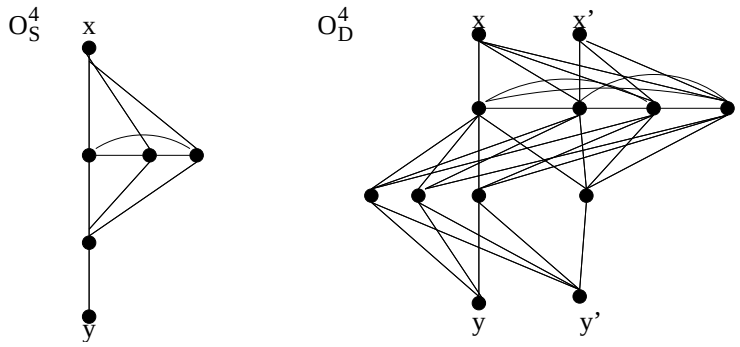

Figure 2.3: Single-Input One-Way Switch O^4

The graph O_D^k has the property that given any set of $m < k-1$ vertices of the form $\{(2,a)|a > 0\}$, there are exactly $k-1-m$ vertices $\{(1,b)|b > 0\}$ such that these vertices form a complete bipartite graph with sets of size m and $k-1-m$. We will use this fact later in describing the strategy of the Duplicator.

In order to prove the first item, we must exhibit a sequence of moves for the Spoiler in the $(\exists, k)$-pebble game on (O_S^k, O_D^k), starting in position $\{xx'\}$, such that either the Duplicator loses or the position $\{yy'\}$ occurs. The Spoiler first places $k-1$ pebbles on $\{(1,a)|a > 0\}$. Since these are all adjacent to the pebble on x, the Duplicator must place $k-1$ pebbles adjacent to x'. Because, in the Spoiler's graph, there is an edge $((1,a),(1,b))$ for all $a \neq b$, these are all adjacent to each other, the Duplicator cannot respond by placing any of these pebbles on the same vertex. The Duplicator is forced to place these pebbles on $\{(1,a)|a > 0\}$. The Spoiler now picks up the pebbles on xx' and places a pebble on $(2,0)$. This pebble is adjacent to every pebble the Spoiler placed in the previous row, so the Duplicator must place his pebble on $(2,0)$. Now the Spoiler picks up any pair of pebbles from row 1 and places a pebble on y. The Duplicator must place a pebble on either y or y', but it must be adjacent the pebble placed in row 2. Since $(2,0)$ is not connected to y, the Duplicator must play on y', arriving at the position $\{yy'\}$.

Now we must show the existence of the two disjoint sets of positions. We first define a set of partial homomorphisms, and then split it to form the Pretrapped and Trapped positions. Given a partial homomorphism $P|O_S^k \to O_D^k$, we define $R_{i,+}(P) = \{(i,a)|\exists u,(u,(i,a)) \in P \wedge a > 0\}$, and $R_{i,0}(P) = \{(i,a)|\exists u,(u,(i,a)) \in P \wedge a = 0\}$. Also, let $T(P) = R_{1,0}(P) \cup R_{2,0}(P)$. Let S be the set of all partial homomorphisms $P|O_S^k \to O_D^k$ such that:

1. $|P| \leq k$

2. $|T(P)| > 0 \Rightarrow xx' \notin P$

3. $\neg(xx' \in P \wedge yy' \in P)$

We say that a position P is Pretrapped if $P \in S$ and $xx' \in P$, and P is Trapped if $P \in S$ and $xx' \notin P$.

Now it remains to prove that the sets of Pretrapped and Trapped positions have the desired properties. By definition, Pretrapped and Trapped positions are partial homomorphisms. Clearly any subset of a Trapped position is Trapped, and any subset of a Pretrapped position is either Trapped or Pretrapped. The position $\{xx'\} \in S$, so $\{xx'\}$ is Pretrapped. The positions $\{yy\}$ and $\{yy'\}$ are in S, so they are Trapped.

Assume that P is Pretrapped; then $|T(P)| = 0$, and every pebble that the Duplicator has placed on row 2 is on the positive side and is therefore connected to y. This means that $P \cup \{yy\}$ is a partial homomorphism, and, since we have not changed any of the criteria for being in S or removed xx', $P \cup \{yy\}$ is Pretrapped.

Assume that P is Trapped; then $xx' \notin P$. There is no vertex adjacent to x' that is not also adjacent to x, so when the Duplicator must play on one of them, it is always acceptable to play on x. This means that $P \cup \{xx\}$ is Trapped.

Now we must show that for every position $P \in S$ such that $|P| < k$, and for every $u \in V(O_S^k)$, there is a $v \in V(O_D^k)$ such that $P \cup \{uv\} \in S$, and if P was Trapped, then so is $P \cup \{uv\}$.

Case 1: If $u = x$, then $v = x$. Clearly P was Trapped, and therefore so is $P \cup \{xx\}$.

Case 2: If $u = (1, a)$, then if $|R_{1,+}(P)| + |R_{2,+}(P)| < k - 1$, then there exists $(1, b)$, $b > 0$, such that $(1, b)$ is adjacent to every vertex in $R_{2,+}(P)$, so choose $v = (1, b)$. This is still a partial homomorphism. It is also still in S because we have not affected the criteria for membership. If $|R_{1,+}(P)| + |R_{2,+}(P)| = k - 1$, then $v = (1, 0)$. Since $|R_{1,+}(P)| + |R_{2,+}(P)| = k - 1$, we know that $|T(P)| = 0$, and $xx' \notin P$. Thus $P \cup \{uv\} \in S$.

Case 3: If $u = (2, 0)$, then if $|R_{1,+}(P)| < k - 1$, then there exists $(2, b)$, $b > 0$, such that $(2, b)$ is adjacent to every vertex in $R_{1,+}(P)$, so choose $v = (2, b)$. This is still a partial homomorphism. It is also still in S because we have not affected the criteria for membership. If $|R_{1,+}(P)| = k - 1$, then $v = (2, 0)$. Since $|R_{1,+}(P)| = k - 1$, we know that $|T(P)| = 0$,

and $xx' \notin P$. Thus $P \cup \{uv\} \in S$.

Case 4: If $u = y$, then, if P is Pretrapped, $P \cup \{yy\}$ is Pretrapped, so $v = y$. If P is Trapped, then check $R_{2,0}(P)$. If $|R_{2,0}(P)| = 0$, then $v = y$, else $v = y'$. □

We call (O_S^k, O_D^k) the Single-Input One-Way Switch.

Corollary 2.1.3 *The Spoiler can reach $\{(y, y')\}$ from $\{(x, x')\}$ in the $(\exists, k)$-pebble game on the Single-Input One-Way Switch, but not in the $(\exists, k-1)$-pebble game on the Single-Input One-Way Switch.*

Proof: The first part of the Corollary is simply a restatement of condition 1 from Lemma 2.1.2. Assume for the sake of contradiction that the Spoiler can reach $\{(x, x')\}$ from $\{(y, y')\}$ in the $(\exists, k-1)$-pebble game on the Single-Input One-Way Switch. Then, from the position $\{(x, x'), (y, y)\}$ of the $(\exists, k)$-pebble game, the Spoiler could reach the position $\{(y, y'), (y, y)\}$ by ignoring the pebbles on (y, y) and playing the $(\exists, k-1)$-pebble game. We know that the position $\{(x, x'), (y, y)\}$ is Pretrapped, but the position $\{(y, y'), (y, y)\}$ is neither Pretrapped nor Trapped. This is a contradiction because the Duplicator can avoid such positions from Pretrapped positions. □

Because of Corollary 2.1.3, the Spoiler has to use all of his pebbles in order to make progress. The "One-Way" aspect of the Switch lies in the fact that $\{(y, y')\}$ is Trapped, and $\{(x, x')\}$ is not. This means that the Duplicator can avoid $\{(x, x')\}$ from $\{(y, y')\}$.

2.1.3 Twisted Switches

The Twisted Switch consists of an H gadget, an I gadget, and two Single-Input One-Way Switches in parallel. Figure 2.4 shows a schematic of the Twisted Switch with boxes where the Single-Input One-Way Switches would be. We use a Twisted Switch in the reduction to initialize the game. The construction of the Twisted Switch is the same as that in Grohe [15], except that we substitute a Single-Input One-Way Switch in place of what Grohe calls a Threshold Switch.

The following Lemma introduces a set of positions of the Single-Input One-Way Switch, called Switched positions. Within the Twisted Switch, the Duplicator uses Switched positions instead of Trapped and Pretrapped positions on the Single-Input One-Way Switch.

Lemma 2.1.4 *There is a set of positions of the $(\exists, k)$-pebble game on O^k, called Switched positions, such that*

1. *$\{xx', yy\}$ and $\{xx, yy'\}$ are Switched.*
2. *If P is Switched, then either $P \cup \{xx', yy\}$ or $P \cup \{xx, yy'\}$ is Switched.*
3. *The Duplicator can avoid positions that are not Switched from Switched positions in the $(\exists, k)$-pebble game.*

Proof: Let the set of Switched positions be the set of all $P \in S$, such that if $yy \in P$, then $P \cup \{xx'\} \in S$.

Clearly $\{xx', yy\}$ and $\{xx, yy'\}$ are Switched. If P is Switched, then either P is Trapped or P is Pretrapped. If P is Pretrapped, then $P \cup \{xx'\}$ is also Pretrapped, because $xx' \in P$, and $P \cup \{yy\}$ is Pretrapped, as proven earlier. Therefore $P \cup \{xx', yy\}$ is Switched. If P is Trapped, then yy is either in P or not. If $yy \in P$, then $P \cup xx' \in S$, and therefore $P \cup \{xx', yy\}$ is Switched. If $yy \notin P$, then $P \cup \{xx, yy'\} \in S$, and $P \cup \{xx, yy'\}$ is Switched.

Based on the earlier proof, if $P \in S$, then the Duplicator can play to some $P' \in S$. We only need to show that the Duplicator can maintain the property that if $yy \in P$, then $P \cup \{xx'\} \in S$. If $P \cup \{xx', yy\}$ is Switched, then the Duplicator plays according to a Trapped strategy on $P \cup \{xx'\}$. If $P \cup \{xx, yy'\}$ is Switched, then the Duplicator plays according to a Trapped strategy on $P \cup \{yy'\}$. □

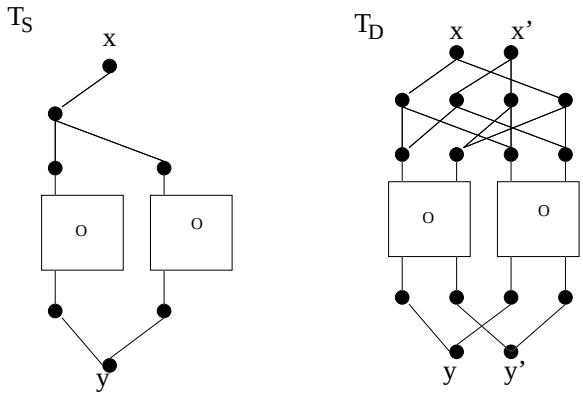

Figure 2.4: Twisted Switch [15]

Lemma 2.1.5 *On the Twisted Switch, the Spoiler can reach $\{yy'\}$ from $\{xx'\}$ in the $(\exists, k)$-pebble game, and there exists a set of positions of the $(\exists, k)$-pebble game called Twisted positions such that*

1. *From Twisted Switches, the Duplicator can avoid non-Twisted positions.*
2. *$\{yy\}$, and $\{yy'\}$, are Twisted positions.*
3. *If P is a Twisted position, then $P \cup \{xx'\}$ is a Twisted position.*

Proof: It is easy to see that the Spoiler can reach yy' from xx'. The Spoiler plays to either ii' or jj' in H, and the traverses one of the One-way switches, and reaches yy'.

Let the Twisted positions be all those positions that meet one of the following criteria:

1. $P_H \subset swi$, $P_I \subset fix_i$, P_{O_l} is Switched, $P_{O_r} \cup \{xx, yy\}$ is Trapped.
2. $P_H \subset swi$, $P_I \subset fix_j$, P_{O_r} is Switched, $P_{O_l} \cup \{xx, yy\}$ is Trapped.
3. $P_H \subset fix$, $P_I \subset fix_i$, P_{O_r} is Switched, $P_{O_l} \cup \{xx, yy\}$ is Trapped.
4. $P_H \subset fix$, $P_I \subset fix_j$, P_{O_l} is Switched, $P_{O_r} \cup \{xx, yy\}$ is Trapped.

Clearly, $\{yy\}$ and $\{yy'\}$ are Twisted. Assume P is twisted. We know that $P_I \subset fix_i$ or $P_I \subset fix_j$. Since xx' is in both of these automorphisms, $P \cup \{xx'\}$ is Twisted.

Given any Twisted position P, it falls in one of the four categories above. On each part of the graph, the Duplicator simply plays according to the appropriate strategy. □

The Twisted Switch will be used to initialize the game. To do this, $x \in T_S$ and $x' \in T_D$ are colored the same color, while $x \in T_D$ is colored a different color. Thus, if the Spoiler plays on x, then the Duplicator must play on x'. From here the Spoiler can play to $\{(y, y')\}$. The Twisted positions allow the Duplicator to avoid $\{(x, x)\}$, which is a losing position.

2.1.4 Reduction from MCV to $(\exists, k)$-Pebble Game

Theorem 2.1.6 *For every fixed $k \geq 2$, determining the winner of the $(\exists, k)$-pebble game is P-Complete.*

Proof: This reduction is modified only slightly from the reduction in [15]. The reduction proceeds as follows: Given a circuit C, an assignment of values to the input nodes, and a node v, we construct a graph C_D. For each node a in the circuit, there are three choices:

1. If a is an input node, then C_D contains vertices a and a'. If the value of a in C is false, then we color a' black.

2. If a is an **and** node with parents b and c, then C_D contains nodes a and a'; a copy of H_D, called H_a with h and h' identified with a and a'; and two copies of O_D^k, one of which has x and x' connected to i and i', and y and y' connected to b and b' called O_{ab}. The other copy connects j and j' to c and c' in a similar manner and is called O_{ac}.

3. If a is an **or** node with parents b and c, then C_D contains nodes a and a'; a copy of I_D called I_a with h and h' identified with a and a'; and two copies of O_D^k, one of which has x and x' connected to i and i', and y and y' connected to b and b' called O_{ab}. The other copy connects j and j' to c and c' in a similar manner and is called O_{ac}.

In any case, a is colored white.

The construction of C_S is similar, except that we do not add a' and we use the Spoiler version of each gadget and switch instead of the Duplicator version.

In addition, there is a Twisted Switch T, such that x is colored a fresh color in C_S, and x' is colored the same color in C_D. Also, in C_S, y in T is connected to v, while in C_D, y is connected to v and y' is connected to v'.

The basic idea is that the Spoiler plays from the top of the Twisted Switch through the graph attempting to reach a false input. Because of the properties of the H and I gadgets and the Single-Input One-Way Switch, the Spoiler can do this if the value of the goal node is false. If the value of the goal node is true, then the Duplicator can play along indefinitely. He does this by choosing a path down the simulated circuit which leads to a true input. If the Spoiler attempts to depart from the intended mode of play, the Duplicator can use the properties

of Trapped and Pretrapped strategies to arrive at a partial homomorphism that is a subset of the identity.

Now we want to prove that if the value of node v in C is false, then the Spoiler wins the $(\exists, k)$-pebble game on (C_S, C_D), and if the value of node v is true, then the Duplicator wins the $(\exists, k)$-pebble game on (C_S, C_D). First, assume that the value of v is false. We must show that the Spoiler wins the $\exists k$-pebble game. First, the Spoiler can play starting on xx', and play to vv'. The proof is by induction on the height of v. Assume that the height of v is 0. This means that v is an input node. Since we assume that the value of v is false, by construction, v and v' are different colors, so the Spoiler wins. Now assume that the height of v is n and for every node u of height less than n, if the value of u is false, then the Spoiler wins. If v is an and node with inputs b and c, then one of the inputs of v is false. Assume, without loss of generality, that b is false. In the $(\exists, k)$-pebble game on (H_S, H_D), the Spoiler can reach ii' from hh'. From this position the Spoiler can reach bb' along O_S^k. By induction, the Spoiler wins the game starting from bb'. The argument is analogous for the or nodes.

Now that we have shown that if the value of v is false, then the Spoiler wins the $(\exists, k)$-pebble game, we must show that if the value of v is true, then the Duplicator wins the $(\exists, k)$-pebble game.

A position P is safe if the position on T is twisted and there exists a node a such that:

1. For every one-way switch O adjacent to aa', $P_O \cup \{yy\}$ is Trapped, and everywhere else P acts as the identity; or

2. For every one-way switch O adjacent to aa', $P_O \cup \{yy'\}$ is Trapped, and

   (a) a is an **and** node with parents b and c, and $P_{H_a} \subseteq swi$, $P_{O_{ab}} \cup \{xx'\}$ is Pretrapped, and $P_{O_{ac}} \cup \{xx'\}$ is Pretrapped.

   (b) a is an **or** node with parents b and c with one of the following:

      i. b is true and $P_{I_a} \subseteq fix_j$, $P_{O_{ab}} \cup \{xx'\}$ is Pretrapped, and $P_{O_{ac}} \subseteq id$.

      ii. c is true and $P_{I_a} \subseteq fix_i$, $P_{O_{ac}} \cup \{xx'\}$ is Pretrapped, and $P_{O_{ab}} \subseteq id$.

   (c) a is an input node with $aa \notin P$.

We now must show that the set of safe positions is a winning strategy for the Duplicator. Any safe position is a partial isomorphism. The position vv' is clearly a safe position. Since

all of the criteria for being safe involve subset properties, any subset of a safe position is safe. Now, given any safe position P, we must show that the Duplicator can play into another safe position. Assume P satisfies case 1. The Duplicator plays according to the identity everywhere except the one-way switches adjacent to a. On one of these subgraphs O, the Duplicator plays according to a Trapped strategy on $O \cup \{yy\}$ if $|P_O| < k-1$, and according to a Trapped strategy on O otherwise.

If P satisfies case 2, then on each one-way switch O adjacent to a, the Duplicator plays according to a Trapped strategy on $P_O \cup \{yy'\}$.

If P satisfies case 2a, then we play a Trapped strategy on $O_{ab} \cup \{xx'\}$ and $O_{ac} \cup \{xx'\}$, unless there are already $k-1$ pebbles placed on one of these subgraphs, in which case we simply play according to a Trapped strategy on that subgraph. As before we play according to the identity everywhere outside the vicinity of a, and according to an automorphism on H_a.

The Duplicator plays similarly for the other cases.

This reduction is clearly in L, because we replace each gate in the circuit with a gadget whose size depends solely on what type of gate it is. We can easily keep track of the indices for the loop in a small amount of space. We have used 10 different colors for the reduction. We use three for the H and I gadgets, four for the rows of the Single-Input One-Way Switches, one for the input of the Twisted Switch, one to encode the values of the circuit inputs, and one base color for everything else.

□

2.2 Complexity of the $(\exists, k)$-Pebble Game with k Part of the Input

Kasai, Adachi and Iwata [17] showed that the following pebble game, which (to avoid confusion in the terminology) we will call the KAI game, is EXPTIME-complete. The game is played between two players, called Player I and Player II. An instance of the KAI game is a quadruple (X, S, R, t), where X is a set of nodes, $R \subseteq X^3$ is a set of *rules*, $S \subseteq X$ is the initial position of the game, and $t \in X$ is the *goal*. There are as many pebbles as nodes in the initial position S; at the start of the game, each node in S has a pebble on it. The two players take turns, and in each turn they slide a pebble as follows: If $(x, y, z) \in R$ is a rule, then the current player may slide a pebble from x to z, if there are pebbles on x and y, but not z. The

first player to place a pebble on the goal t wins. The problem is, given (X, R, S, t), does Player I have a winning strategy?

We will show that the $(\exists, k)$-pebble game is EXPTIME-complete by exhibiting a reduction from the KAI game. The reduction proceeds by constructing an instance of the $(\exists, k)$-pebble game such that the Spoiler and the Duplicator simulate the instance of the KAI game. In particular, the Spoiler can win only by simulating a winning strategy for Player I in the KAI game. If there is no winning strategy, then the Spoiler does not gain any advantage by not simulating the KAI game. For this reduction, we use $k = |X|$.

To perform this simulation, we use a Twisted Switch (Section 2.1.3) to initialize the game, and new switches to allow Player I and Player II to choose rules, to apply the rules, and to force the Spoiler to simulate the game.

2.2.1 The Gadgets H^m and I^m

These gadgets allow the Spoiler and Duplicator to choose rules of the KAI game to use. In both the H^m and I^m gadgets, the nodes y^i, y^i_0, y^i_1 are colored the same color, but the color of y^i is different from the color of y^j for $i \neq j$. See Figures 2.5 and 2.6 for examples of the H^m and I^m gadgets respectively.

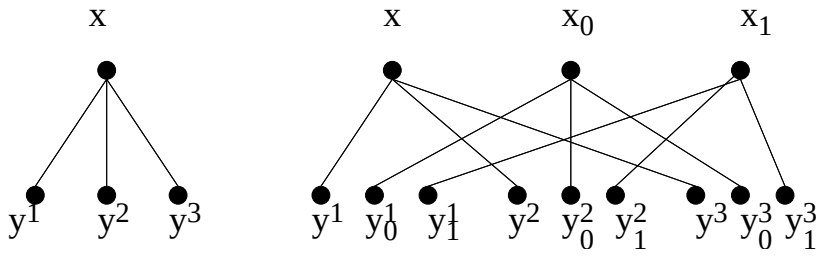

Figure 2.5: H^3 Gadget

Lemma 2.2.1 *For every $k \geq 2$, in the $(\exists, k)$-pebble game on (H^m_S, H^m_D), from a position $\{xx_j\}$, $j \in \{0, 1\}$, the Spoiler can reach $\{y^i y^i_j\}$ for any i.*

Proof: Clear based on vertex coloring. □

Lemma 2.2.2 *For every $k \geq 2$, in the $(\exists, k)$-pebble game on I^m_S, I^m_D, from a position $\{xx_j\}$, $j \in \{0, 1\}$, the Duplicator can choose any $1 \leq i \leq m$, and avoid $\{y^l y^l_j\}$ for $l \neq i$.*

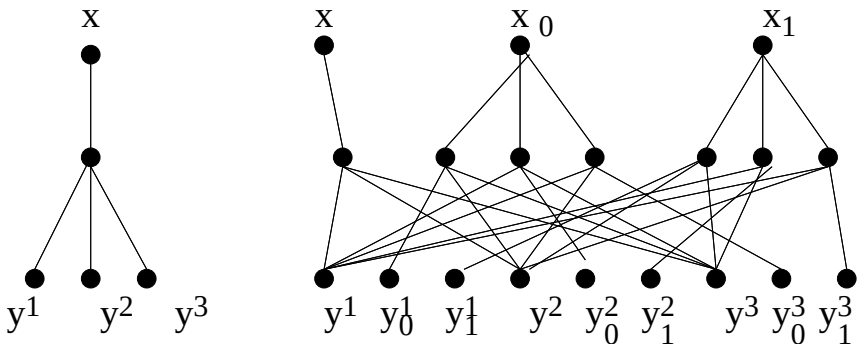

Figure 2.6: I^3 Gadget

Proof: In a similar way to that used on the smaller I gadget, the Duplicator chooses an intermediate step which forces the Spoiler to play a certain way in order to carry information forward in the game. □

2.2.2 Multiple-Input One-Way Switches for the $(\exists, k)$-Pebble Game

The idea of the Multiple-Input One-Way Switch is to restrict the Spoiler's potential winning strategies. We simulate each node x^i in the KAI game by using three nodes in the Duplicator's graph, x_0^i, x_1^i, x^i. These correspond to not having a pebble on x^i in the simulated game, having a pebble on x^i in the simulated game, and no information about x^i, respectively. In the Multiple-Input One-Way Switch, the Spoiler can make progress only if he has information about each node in the simulated game. Also, if the Spoiler attempts to play backwards through the Switch, he will end up with no information about any nodes in the simulated game.

Lemma 2.2.3 *For every $k \geq 2$, there exists a pair of graphs M_S^k, and M_D^k such that:*

$$\{x^1, x_0^1, x_1^1, \ldots, x^{k-1}, x_0^{k-1}, x_1^{k-1}, y^1, y_0^1, y_1^1, \ldots, y^{k-1}, y_0^{k-1}, y_1^{k-1}\} \subset V(M_D^k),$$

$\{x^1, \ldots, x^{k-1}, y^1, \ldots, y^{k-1}\} \subset V(M_S^k)$, *$|V(M_S^k)|$ is $O(k^2)$, $|V(M_D^k)|$ is $O(k^2)$, and the following properties hold:*

1. *From a position $\{x^i x_{j_i}^i | 1 \leq i \leq k-1, j_i \in \{0,1\}\}$, the Spoiler can reach the position $\{y^i y_{j_i}^i | 1 \leq i \leq k-1, j_i \in \{0,1\}\}$ in the $(\exists, k)$-pebble game on M_S^k and M_D^k.*
2. *There exist two disjoint sets of positions of the $(\exists, k)$-pebble game on (M_S^k, M_D^k), called Pretrapped and Trapped positions, such that:*

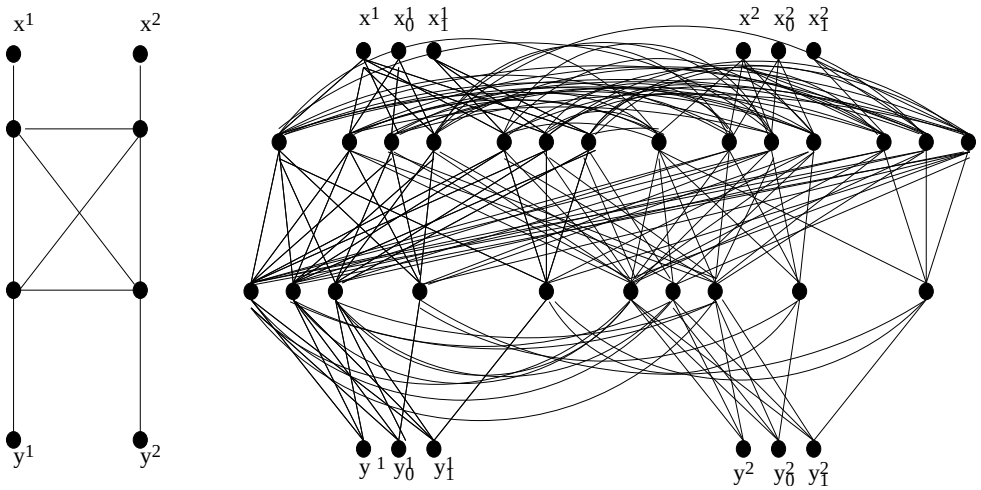

Figure 2.7: A Subgraph of M^4

(a) Pretrapped and Trapped positions are partial homomorphisms.

(b) From Pretrapped positions, the Duplicator can avoid positions that are not Pretrapped and not Trapped.

(c) From Trapped positions, the Duplicator can avoid positions that are not Trapped.

(d) From any position $P = \{x^i a | 1 \leq i \leq k-1\}$ *where* $|\{x^i x^i_j \in P | j \in \{0,1\}\}| < k-1$*, the Duplicator can avoid* $y^i y^i_j$ *for all* $1 \leq i \leq k-1, j \in \{0,1\}$*.*

(e) All positions that are subsets of positions of the form $\{x^i x^i_{j_i} | 1 \leq i \leq k-1, j_i \in \{0,1\}\}$*, are Pretrapped.*

(f) If P *is Pretrapped and* $|P| < k$ *, then* $P \cup \{y^i y^i\}$ *is Pretrapped for all* i*.*

(g) Any position in which all of the Spoiler's pebbles are on nodes y^i*, is Trapped.*

(h) If P *is Trapped and* $|P| < k$*, then* $P \cup \{x^i x^i\}$ *is Trapped for all* i*.*

Proof: First, we define M^k_S and M^k_D. Then, we show that these graphs have the desired properties.

$$
\begin{aligned}
V(M_D^k) &= \{x^1, x_0^1, x_1^1, \ldots, x^{k-1}, x_0^{k-1}, x_1^{k-1}, y^1, y_0^1, y_1^1, \ldots, y^{k-1}, y_0^{k-1}, y_1^{k-1}\} \cup \\
&\quad \{A_j^i | 1 \le i \le k-1, -(k-1) \le j \le k-1\} \cup \\
&\quad \{B_j^i | 1 \le i \le k-1, -(k-1) \le j \le 1\} \\
E(M_D^k) &= \{(x^i, A_j^i) | 1 \le i \le k-1, -(k-1) \le j \le k-1\} \cup \\
&\quad \{(x_0^i, A_j^i) | 1 \le i \le k-1, -(k-1) \le j \le -1\} \cup \\
&\quad \{(x_1^i, A_j^i) | 1 \le i \le k-1, 1 \le j \le k-1\} \cup \\
&\quad \{(A_j^i, A_m^l) | i \ne l, |j| \ne |m|\} \cup \\
&\quad \{(B_j^i, B_m^l) | i \ne l, (j < 0 \wedge m < 0) \vee (j \ge 0 \wedge m \ge 0)\} \cup \\
&\quad \{(A_j^i, B_m^l) | j \ne 0, -(k-1) \le m \le -1, |j| \ne ((l+m) \bmod k) + 1\} \cup \\
&\quad \{(A_0^i, B_m^l)\} \cup \\
&\quad \{(A_j^i, B_0^l) | (j < 0) \vee (i \ne l)\} \cup \\
&\quad \{(A_j^i, B_1^l) | (j > 0) \vee (i \ne l)\} \cup \\
&\quad \{(B_j^i, y^i) | 1 \le i \le k-1, -(k-1) \le j \le -1\} \cup \\
&\quad \{(B_j^i, y_j^i) | 1 \le i \le k-1, j \in \{0, 1\}\} \cup \\
&\quad \{(B_j^i, y_k^i) | 1 \le i \le k-1, -(k-1) \le j \le -1, k \in \{0, 1\}\}
\end{aligned}
$$

$$
\begin{aligned}
V(M_S^k) &= \{x^1, \ldots, x^{k-1}, y^1, \ldots, y^{k-1}\} \cup \\
&\quad \{A^i | 1 \le i \le k-1\} \cup \\
&\quad \{B^i | 1 \le i \le k-1\} \\
E(M_S^k) &= \{(x^i, A^i)\} \cup \\
&\quad \{(A^i, A^j) | i \ne j\} \cup \\
&\quad \{(A^i, B^j)\} \cup \\
&\quad \{(B^i, B^j) | i \ne j\} \cup \\
&\quad \{(B^i, y^i)\}
\end{aligned}
$$

One can picture this graph as a set of connected switches, each of which is similar to the Single-Input One-Way Switch. In particular, each individual switch needs to be colored

a separate color. Figure 2.7 shows two component switches for the game on 4 pebbles. There would be one more in the full Multiple-Input One-Way Switch, connected to each of these two in the same way.

From a position $P = \{x^i x^i_{j_i} | 1 \leq i \leq k-1, j_i \in \{0,1\}\}$, the Spoiler can take his extra pebble and place it on A^1. We know that x^1_0 is only connected to A^1_i where $i < 0$, and x^1_1 is only connected to A^1_i where $i > 0$. So, if $x^1 x^1_0 \in P$, the Duplicator must place a pebble on some A^1_i, $i < 0$, and if $x^1 x^1_1 \in P$, the Duplicator must play on Q^1_i, $i > 0$. Now, the Spoiler picks up the pebble on x^1 and places it on A^2. Similarly, the Duplicator must play negative if her pebble is on x^2_0 and positive if her pebble is on x^2_1. The Duplicator's choice is more limited than in the previous move, because the Duplicator can only play on A^2_j such that $|j| \neq |i|$. After $k-1$ such moves, P' will consist of $\{x^{k-1} x^{k-1}_{j_{k-1}}\}$ union a set $\{A^i A^i_{l_i} | 1 \leq i < k-1\}$, such that $|l_i| \neq |l_j|$ for all $i \neq j$, and the sign of l_i matches the sign of j_i for all i.

The Spoiler next picks up the pebble on x^{k-1} and places it on B^1. For any $B^l_m \in M^k_D$, $m < 0$, there is a j, $|j| > 0$, such that $(A^i_j, B^l_m) \notin E(M^k_D)$ for any i. This means that the Duplicator must place a pebble on either B^1_0, or B^1_1 depending on the sign of A^1_j. Next, the Spoiler picks of the pebble on A^1 and places it on B^2. The Duplicator must respond by playing a pebble on B^2_0 or B^2_1, depending on the sign of A^2_j. This sequence is continued.

From here it is easy to see that the Spoiler can reach $\{y^i y^i_{j_i} | 1 \leq i \leq k-1, j_i \in \{0,1\}\}$.

Now we must exhibit a set of positions with the appropriate properties.

Let S be the set of all partial homomorphisms, P, from M_S to M_D such that:

1. $\exists i, j$ such that $x^j_i \in Im(P) \Rightarrow \nexists l, m$ such that $B^l_m \in Im(P)$.
2. $\forall i$, $y^i_0 \in Im(P) \Rightarrow \nexists j < 0$ such that $a^i_j \in Im(P)$.
3. $\forall i$, $y^i_1 \in Im(P) \Rightarrow \nexists j > 0$ such that $a^i_j \in Im(P)$.

P is Pretrapped if $\exists i, j$ such that $x^j_i \in Im(P)$ and $P \in S$.

P is Trapped if $P \in S$ and P is not Pretrapped.

The last four properties of these sets of positions are easy to see. The difficulty arises in proving that given any Trapped position, the Duplicator can play so that the position remains Trapped, and given any Pretrapped position, the Duplicator can play so that the position is either Trapped or Pretrapped. We prove that given any position in S, the Duplicator can play so that

the new position is also in S and if the previous position was Trapped, then the new position is Trapped. Given a position in S, there are four basic types of moves that the Spoiler can make.

Case 1: $u = x^i$. The Duplicator can always play on x^i.

Case 2: $u = A^i$. If $x^i x^i_j \notin P$ for any j, then the Duplicator can play on A^i. Assume $x^i x^i_0 \in P$ (the case of $x^i x^i_1 \in P$ is symmetric). We need to find node A^i_j, such that $j < 0$, and A^i_j is connected to all of the chosen nodes on the A and B levels. Since there are strictly fewer than $k-1$ such nodes, there is a choice such that the partial homomorphism property is maintained.

Case 3: $u = B^i$. If $\exists l, m$ such that $x^l_m \in Im(P)$, then $\exists j < 0$ such that $P \cup \{(b^i, b^i_j)\}$ is a partial homomorphism. If $\exists l, m \geq 0$ such that $b^l_m \in Im(P)$, then either b^i_0 or b^i_1 will maintain the partial homomorphism. Let $A = \{a^l_m \in Im(P) | m \neq 0\}$. If $|A| = k-1$, then $\exists j \neq 0$ such that $a^i_j \in A$. Then, if $j > 0$, the Duplicator plays b^i_1, otherwise the Duplicator plays b^i_0.

Case 4: $u = y^i$: If P is Pretrapped, then the Duplicator can play y^i. If P is Trapped, then the Duplicator can play one of y^i, y^i_0, y^i_1.

Now we show that using this strategy, the Duplicator will avoid $y^i y^i_j$ for all $1 \leq i \leq k-1, j \in \{0,1\}$ from any position $P = \{x^i a | 1 \leq i \leq k-1\}$ where $|\{x^i x^i_j \in P | j \in \{0,1\}\}| < k-1$. We know that P is Pretrapped, so $P \cup \{y^i y^i_j | 1 \leq i \leq k-1, j \in \{0,1\}$ is Pretrapped. For sake of contradiction, assume that the Spoiler can play to $\{y^i y^i_j\}$ for some i, j. This means that the Duplicator would have played a pebble on B^i_l for some $l \geq 0$. There are two reasons that the Duplicator will play a pebble on B^i_l with $l \geq 0$: Either there are i' and l' such that $B^{i'}_{l'}$ is in $Im(P')$; or P' contains $k-1$ pairs of the form $A^q A^q_m$ with $m \neq 0$. In the first case we can recurse to the second case. Now it is easy to see that the Duplicator would never actually play to this position, because for some i, the pair $x^i x^i \in P$. If the Spoiler plays on A^i, the Duplicator will play on A^i_0.

□

2.2.3 The Rule Gadget

The rule gadgets are used to simulate a move of the KAI game. One rule gadget causes the Spoiler to lose if the rule gadget corresponds to a rule that cannot be applied, and another causes the Duplicator to lose if the rule cannot be applied.

First, we define the rule gadget RS^n, shown in Figure 2.8, that is used to keep the Spoiler playing nicely.

$$
\begin{aligned}
V(RS^n_D) &= \{x, x', x_0, x'_0, x_1, x'_1, y, y', y_0, y'_0, y_1, y'_1, z, z', z_0, z'_0, z_1, z'_1\} \cup \\
&\quad \{x^i, x^{i\prime}, x^i_0, x^{i\prime}_0, x^i_1, x^{i\prime}_1 | 1 \leq i < n-3\} \cup \\
&\quad \{a, a_x, a_y, a_z\} \\
E(RS^n_D) &= \{(x^i, x^{i\prime}), (x^i_0, x^{i\prime}_0), (x^i_1, x^{i\prime}_1) | 1 \leq i < n-3\} \cup \\
&\quad \{(x, a_x), (x, a_y), (x, a_z), (x_0, a_x), (x_0, a_y), (x_0, a_z)\} \cup \\
&\quad \{(x_1, a_y), (x_1, a), (x_1, a_z)\} \cup \\
&\quad \{(y, a_x), (y, a_y), (y, a_z), (y_0, a_x), (y_0, a_y), (y_0, a_z)\} \cup \\
&\quad \{(y_1, a_x), (y_1, a), (y_1, a_z)\} \cup \\
&\quad \{(z, a_x), (z, a_y), (z, a_z), (z_0, a_x), (z_0, a_y), (z_0, a)\} \cup \\
&\quad \{(z_1, a_x), (z_1, a_y), (z_1, a_z)\} \cup \\
&\quad \{(a, x'_0), (a, y'_1), (a, z'_1)\} \cup \\
&\quad \{(q, x'), (q, y'), (q, z') | q \in \{a_x, a_y, a_z\}\}
\end{aligned}
$$

$$
\begin{aligned}
V(RS^n_S) &= \{x, x', y, y', z, z'\} \cup \\
&\quad \{x^i, x^{i\prime} | 1 \leq i < n-3\} \cup \\
&\quad \{a\} \\
E(RS^n_S) &= \{(x^i, x^{i\prime}) | 1 \leq i < n-3\} \cup \\
&\quad \{(x, a), (y, a), (z, a), (a, x'), (a, y'), (a, z')\}
\end{aligned}
$$

Lemma 2.2.4 *If the rule gadget RS^n does not correspond to a legal rule, that is, if one of xx_1, yy_1, zz_0 is not in P, then the Duplicator can avoid $z'z'_1$ in the $(\exists, k)$-pebble game on RS^n.*

Proof: Each rule gadget corresponds to a rule $R(x, y, z)$. If $\{(x, x_1), (y, y_1), (z, z_0)\} \subseteq P$, then the Spoiler can reach $\{(x', x'_0), (y', y'_1), (z', z'_1)\}$. Otherwise, if the Spoiler plays on x', y', z', the Duplicator is forced to play on x', y', z'. □

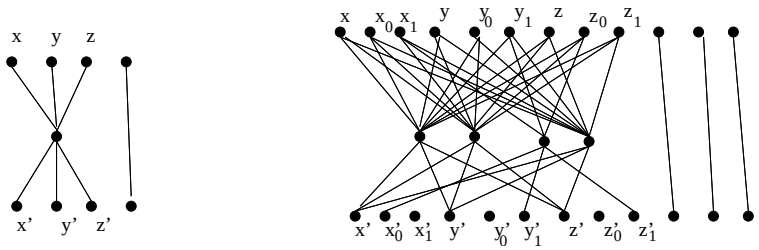

Figure 2.8: The rule gadget RS^n that penalizes the Spoiler for choosing a bad rule

Next, we define the gadget RD^n, shown in Figure 2.9.

$$
\begin{aligned}
V(RD^n_D) &= \{x, x', x_0, x_0', x_1, x_1', y, y', y_0, y_0', y_1, y_1', z, z', z_0, z_0', z_1, z_1'\} \cup \\
&\quad \{x^i, x^{i\prime}, x_0^i, x_0^{i\prime} x_1^i, x_1^{i\prime} | 1 \le i < n-3\} \cup \\
&\quad \{a, a_x, a_y, a_z, a_{xyz}\} \\
E(RD^n_D) &= \{(x^i, x^{i\prime}), (x_0^i, x_0^{i\prime}), (x_1^i, x_1^{i\prime}) | 1 \le i < n-3\} \cup \\
&\quad \{(x, a_x), (x, a_y), (x, a_z), (x_0, a_x), (x_0, a_y), (x_0, a_z)\} \cup \\
&\quad \{(x_1, a_y), (x_1, a), (x_1, a_z), (x, a_{xyz})\} \cup \\
&\quad \{(y, a_x), (y, a_y), (y, a_z), (y_0, a_x), (y_0, a_y), (y_0, a_z)\} \cup \\
&\quad \{(y_1, a_x), (y_1, a), (y_1, a_z), (y, a_{xyz})\} \cup \\
&\quad \{(z, a_x), (z, a_y), (z, a_z), (z_0, a_x), (z_0, a_y), (z_0, a)\} \cup \\
&\quad \{(z_1, a_x), (z_1, a_y), (z_1, a_z)\}, (z, a_{xyz}) \cup \\
&\quad \{(a, x_0'), (a, y_1'), (a, z_1')\} \cup \\
&\quad \{(q, x'), (q, y'), (q, z_0') | q \in \{a_x, a_y, a_z\}\} \cup \\
&\quad \{(a_{xyz}, x'), (a_{xyz}, y'), (a_{xyz}, z')\}
\end{aligned}
$$

$$
\begin{aligned}
V(RD^n_S) &= \{x, x', y, y', z, z'\} \cup \\
&\quad \{x^i, x^{i'} | 1 \leq i < n-3\} \cup \\
&\quad \{a\} \\
E(RD^n_S) &= \{(x^i, x^{i'}) | 1 \leq i < n-3\} \cup \\
&\quad \{(x,a), (y,a), (z,a), (a,x'), (a,y'), (a,z')\}
\end{aligned}
$$

Lemma 2.2.5 *If the rule gadget RD^n does not correspond to a legal rule, that is, if one of xx_1, yy_1, zz_0 is not in P, then the Spoiler can play to $z'z'_0$, which causes the Duplicator to lose.*

Proof: If $\{(x,x_1),(y,y_1),(z,z_0)\} \subseteq P$, then the Spoiler can play to reach the position $\{(x',x'_0),(y',y'_1),(z',z'_1)\}$. If $\{(x,x),(y,y),(z,z)\} \subseteq P$, then the Duplicator can play to $\{(x',x'),(y',y'),(z',z')\}$. Otherwise, the Spoiler can play to $\{(x',x'),(y',y'),(z',z'_0)\}$ which causes the Duplicator to lose because z'_0 is colored a distinct color. □

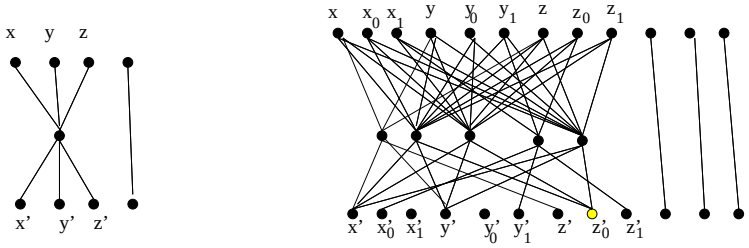

Figure 2.9: The rule gadget RD^n that penalizes the Duplicator for choosing a bad rule

2.2.4 Winning the Game

For each gadget, we define its *boundary* to be the set of nodes which are used to connect that gadget to other gadgets. For M^k, H^m, and I^m, the x and y nodes form the boundary. For RS^n and RD^n, the boundary consists of all the nodes except the middle nodes. For the Twisted Switch, the boundary consists of y in T_S and y, y' in T_D. In the Twisted Switch, x and x' are never connected to any other gadgets.

Lemma 2.2.6 *For each gadget other than the Twisted Switch, starting from a position that is a subset of the identity on the boundary, the Duplicator can avoid any position that is not a subset of the identity on the boundary.*

Proof: For H^m, I^m, RS^n and RD^n this is clear, because for each X in this set, $X_S \subset X_D$. On the Multiple-Input One-Way Switch, this follows from the properties proven in Lemma 2.2.3. □

By combining this lemma with the properties of the Multiple-Input One-Way Switch, we obtain a sufficient condition for the Duplicator to win the $(\exists, k)$-pebble game.

2.2.5 Reduction from KAI Game to $(\exists, k)$-Pebble Game

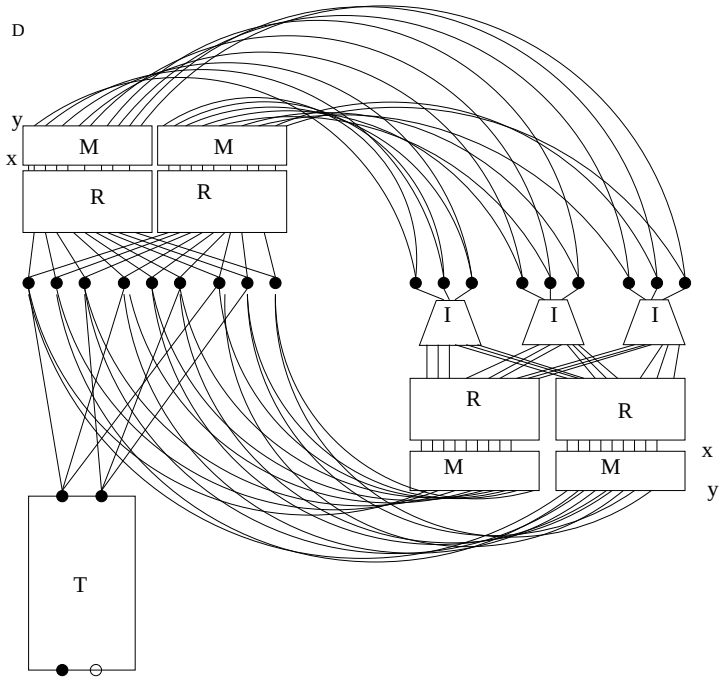

Figure 2.10: Component decomposition of the Duplicator's graph for the reduction

Theorem 2.2.7 *Determining the winner of the $(\exists, k)$-pebble game with k part of the input is EXPTIME-Complete.*

Proof: We perform a reduction from the KAI Game to the $(\exists, k)$-pebble game. An instance of the KAI Game consists of (X, S, R, t). Given this, we form an instance of the $(\exists, k)$-pebble game as follows. First, $k = |X| + 1$, so the Spoiler and Duplicator will be playing with as many pebbles as there are nodes in the KAI game.

The Duplicator's graph and the Spoiler's graph each have two sides. One side represents Player I's turn in the KAI game, while the other side represents Player II's turn.

First, we build Player I's side of the graph. For each $x^i \in X$, we form three nodes in D, called xs^i, xs^i_0, xs^i_1. These three nodes correspond to specific information about the simulated KAI game. If there is a pebble on xs^i_1, then there is a pebble on x in the KAI game, and xs^i_0 corresponds to no pebble on x. A pebble on xs in the Duplicator's graph means that the Spoiler has made a mistake. For each $(x, y, z) \in R$, construct a rule gadget to penalize the Spoiler, connected to each xs, xs_0, xs_1 by H^m gadgets. The other end is then connected to a Multiple-Input One-Way Switch.

On the other half of the graph is another set of nodes, xd^i, xd^i_0, xd^i_1. Connecting the xd nodes to Duplicator punishing rule gadgets is a set of I^m gadgets where $m = |R|$. Given any two of these I^m gadgets A and B, there is an edge between $A.y^i_j$ and $B.y^i_k$ for all i, j, k, and an edge between $A.y^i$ and $B.y^i$ for all i. We then connect a Multiple-Input One-Way Switch to each of the rule gadgets in the obvious way. The outputs of these are connected back around to the first set of state nodes.

We use a Twisted Switch to set up the initial positions. If $x^i \in S$, then there is an edge from y' of the Twisted Switch to xs^i_1, otherwise there is an edge from y' to xs^i_0. There is an edge from y to xs^i for every i. We then color x' a unique color.

The Spoiler's graph is constructed in a similar way. The Spoiler gets xs^i and xd^i for every $x^i \in X$. Also, for each rule and each side, there is a corresponding Multiple-Input One-Way Switch followed by a rule gadget. For the Twisted Switch, we color x the same color that we colored x' in the Duplicator's graph.

The idea is that the two players now simulate the KAI game by playing the $(\exists, k)$-pebble game on these structures. The Spoiler initializes the game by playing on the Twisted Switch. From here, the Spoiler uses the H^m gadgets to choose a rule, then plays through

the RS^n gadget and the Multiple-Input One-Way Switch to simulate the move of Player I in the KAI game. Then the Spoiler continues to play through Player II's side of the structures, allowing the Duplicator to choose a rule to simulate, and applying that rule. If Player I has a winning strategy for the KAI game, then this simulation process will eventually simulate Player I's placing a pebble on t. Because of the coloring of the structures, this causes the Spoiler to win the $(\exists, k)$-pebble game. The more difficult step is showing that if Player I does not have a winning strategy, then the Duplicator wins the $(\exists, k)$-pebble game. If the Spoiler plays nicely and simulates the KAI game, then the Duplicator can avoid t, by playing a smart strategy for the KAI game. If the Spoiler does not play nicely, and departs from the simulation, then, because of the properties of the gadgets, the Duplicator can play along indefinitely.

First, we show that if Player I has a forced win in the KAI game, then the Spoiler can win the $(\exists, k)$-pebble game. A forced win consists of a strategy such that on Player I's turn, there is a legal rule such that for every choice Player II makes, Player I arrives back at a position with a forced win.

If the Spoiler plays on x in the Twisted Switch, the Duplicator must play on x'. This allows the Spoiler to play the initial position of the KAI game. Since we know that there is a rule that corresponds to a forced-win strategy for Player I in the KAI game, the Spoiler can use the H^m gadgets to choose that rule. Then, the Spoiler can traverse the Multiple-Input One-Way Switch, maintaining the values of the nodes. Since this rule was chosen from a forced-win strategy for Player I, we know that it is a legal rule to apply, so the Spoiler can play to the xd with the value of xd corresponding to the simulated game after the chosen rule has been applied.

Now the Spoiler continues play along the I^m gadgets. We know that the Spoiler can reach some rule along these gadgets, but the Duplicator gets to pick which one. The Duplicator must pick a single rule because of the connections among the outputs of the I^m gadgets. This means that the Spoiler can reach the Multiple-Input One-Way Switch that corresponds to some rule, with all of the information intact. Now the Spoiler plays through the Multiple-Input One-Way Switch and the rule gadget to return to the xs side of the structure. Since the Spoiler is playing according to a forced-win strategy, there is a forced-win strategy from this position.

Eventually the Spoiler will place a pebble on td, and the Duplicator will be forced to place a pebble on td_1. Since these are different colors, the Spoiler wins.

Now, assuming that Player I does not have a forced win in the KAI game, we must

exhibit a winning strategy for the Duplicator.

Consider a configuration $C = (O, r)$ of the KAI game where O is the set of nodes that have pebbles on them and r is a Boolean representing which player's turn it is. We call a configuration of the KAI game critical for a position P of the $(\exists, k)$-pebble game if one of the following is true:

1. The number of active nodes is $|S|$, it is the Spoiler's turn, and

   (a) For each Multiple-Input One-Way Switch MD^j on Player II's side, $P|_{MD^j} \cup \{y^i y_1^i | x^i \in O\} \cup \{y^i y_0^i | x^i \notin O\}$ is Trapped.

   (b) For each Multiple-Input One-Way Switch MS^j on Player I's side that corresponds to a rule (x, y, z) such that $x, y \in O$ and $z \notin O$, $P|_{MS^j} \cup \{x^i x_1^i | x^i \in O \setminus \{x\} \cup \{z\}\} \cup \{x^i x_0^i | x^i \notin O \setminus \{x\} \cup \{z\}\}$ is Pretrapped.

   (c) For each Multiple-Input One-Way Switch MS^j on Player I's side that corresponds to a rule (x, y, z) such that either $x \notin O$, $y \notin O$, or $z \in O$, then $P|_{MS^j} \cup \{y^i y_1^i | x^i \in O \setminus \{x, y, z\}\} \cup \{y^i y_0^i | x^i \notin O \cup \{x, y, z\}\} \cup \{y^i y^i | x^i \in \{x, y, z\}\}$ is Pretrapped.

2. The number of active nodes is less than $|S|$, it is the Spoiler's turn and there exists x^k such that:

   (a) For each Multiple-Input One-Way Switch MD^j on Player II's side, $P|_{MD^j} \cup \{y^i y_1^i | x^i \in O\}$ is Trapped, $x^k \notin O$, $P|_{MD^j} \cup \{x^k x^k\}$ is Trapped.

   (b) For each Multiple-Input One-Way Switch MS^j on Player I's side that corresponds to a rule (x, y, z) such that $x, y \in O$ and $z \notin O$, $P|_{MS^j} \cup \{x^i x_1^i | x^i \in O \setminus \{x\}\}$ is Pretrapped, and $x_k \notin O$, $P|_{MD^j} \cup \{x^k x^k\}$ is Pretrapped.

   (c) For each Multiple-Input One-Way Switch MS^j on Player I's side that corresponds to a rule (x, y, z) such that either $x \notin O$, $y \notin O$, or $z \in O$, then $P|_{MS^j} \cup \{y^i y_1^i | x^i \in O \setminus \{x, y, z\}\} \cup \{y^i y^i | x^i \in \{x, y, z\}\}$ is Pretrapped.

3. The number of active nodes is $|S|$, it is the Duplicator's turn, and

   (a) For each Multiple-Input One-Way Switch MS^j on Player I's side, we have that $P|_{MS^j} \cup \{y^i y_1^i | x^i \in O\} \cup \{y^i y_0^i | x^i \notin O\}$ is Trapped.

(b) For each Multiple-Input One-Way Switch MD^j on Player II's side that corresponds to a rule (x, y, z) such that $x, y \in O$ and $z \notin O$, $P|_{MD^j} \cup \{x^i x_1^i | x^i \in O \setminus \{x\} \cup \{z\}\} \cup \{x^i x_0^i | x^i \notin O \setminus \{x\} \cup \{z\}\}$ is Pretrapped.

(c) For each Multiple-Input One-Way Switch MD^j on Player II's side that corresponds to a rule (x, y, z) such that either $x \notin O$, $y \notin O$, or $z \in O$, then $P|_{MD^j} \cup \{y^i y_1^i | x^i \in O \setminus \{x, y, z\}\} \cup \{y^i y_0^i | x^i \notin O \cup \{x, y, z\}\} \cup \{y^i y^i | x^i \in \{x, y, z\}\}$ is Pretrapped.

4. The number of active nodes is less than $|S|$, it is the Duplicator's turn, and there exists x^k such that:

(a) For each Multiple-Input One-Way Switch MS^j on Player I's side, we have that $P|_{MS^j} \cup \{y^i y_1^i | x^i \in O\}$ is Trapped, $x^k \notin O$, $P|_{MS^j} \cup \{x^k x^k\}$ is Trapped.

(b) For each Multiple-Input One-Way Switch MD^j on Player II's side that corresponds to a rule (x, y, z) such that $x, y \in O$ and $z \notin O$, $P|_{MD^j} \cup \{x^i x_1^i | x^i \in O \setminus \{x\}\}$ is Pretrapped, and $x_k \notin O$, $P|_{MD^j} \cup \{x^k x^k\}$ is Pretrapped.

(c) For each Multiple-Input One-Way Switch MD^j on Player II's side that corresponds to a rule (x, y, z) such that either $x \notin O$, $y \notin O$, or $z \in O$, then $P|_{MD^j} \cup \{y^i y_1^i | x^i \in O \setminus \{x, y, z\}\} \cup \{y^i y^i | x^i \in \{x, y, z\}\}$ is Pretrapped.

Let the safe positions be the set of partial homomorphisms P such that $P|_T$ is Twisted and there is a configuration of the KAI game that is not on a winning path for Player I, which is critical for P. Notice that any position where the empty configuration is critical is a winning position for the Duplicator.

What we must show is that for any safe position, the Duplicator can play in such a way as to avoid non-safe positions. The empty position is safe. As an example, consider a safe position P in which the configuration $(O, 1)$ is critical for P. If the Spoiler plays in any MD^j, then the Duplicator plays according to a Trapped strategy on MD^j. If the Spoiler plays on T, then the Duplicator plays according to a Twisted strategy on T. If the Spoiler plays on RS^i, the Duplicator has no choices. If the Spoiler plays on MS^j, then the Duplicator plays according to a Pretrapped strategy on MS^j. If $|O| < |S|$, then, as proven earlier, the Duplicator can avoid $y^i y_j^i$ on all MS^j. Also, if MS^j corresponds to a rule that is illegal in configuration O, then the Duplicator can also avoid $y^i y_j^i$ for all MS^j.

The Duplicator plays similarly on the other side of the graph when a configuration $(O, 2)$ is critical for P.

□

As pointed out in Section 2.1, the structures used in the reduction of MCV to the $(\exists, k)$-pebble game with fixed k were undirected graphs with a fixed number of colors (ten). In contrast, the structures used in the preceding reduction of the KAI game to the $(\exists, k)$-game with k part of the input are undirected graphs with a number of colors that is linear in the size of the input. It is an interesting technical problem to exhibit a reduction of the KAI game to the $(\exists, k)$-game with k part of the input in which the structures are undirected graphs with a fixed number of colors.

2.3 A Framework for Pebble Games

Combinatorial games have been used to analyze logical definability and the expressive power of various logics. Usually a logic L will be parameterized based on some syntactic property k, often number of variables or alternation of quantifiers. This gives rise to logic fragments $L(k)$. With each fragment, we associate a game $G(k)$. This game is played on two structures $\mathbf{A}$ and $\mathbf{B}$, by two players known as Spoiler and Duplicator. The game is designed such that Duplicator has a winning strategy if and only if $\mathbf{A}$ and $\mathbf{B}$ satisfy the same sentences in $L(k)$.

As discussed earlier, for $\exists L^{\omega}_{\infty\omega}$ the corresponding game is the $(\exists, k)$-pebble game. For the first-order logic FO, each fragment $FO(k)$ is the set of first-order sentences with quantifier rank that is at most k, and the game $G(k)$ is the k-move Ehrenfeucht-Fraïssé game. For the infinitary logic $L^{\omega}_{\infty\omega}$ with finitely many variables, each fragment is $L^{k}_{\infty\omega}$, the infinitary logic with k variables. The corresponding game is the k-pebble game, which has been used to study fixed-points logics in finite model theory. (For a survey, see [15].)

For each of these games $G(k)$, we have the decision problem of determining the winner of the game: Given two structures **A**and **B**, does the Duplicator have a winning strategy for $G(k)$ on **A**and **B**? It is easy to show that determining the winner of the the k-move Ehrenfeucht-Fraïssé game is in LOGSPACE. Determining the winner of the k-pebble game is comparatively more difficult. Grohe showed in [15] that determining the winner of the k-pebble game is

complete for polynomial time.

A natural extension of this is to consider the complexity of these games when the parameter k is part of the input. Pezzoli showed in [25] that the complexity of the k-move Ehrenfeucht-Fraïssé game is PSPACE-complete if k is part of the input. This exponential jump in complexity when k is allowed to vary in the Ehrenfeucht-Fraïssé game raises the question, what happens in other games when the parameter is allowed to vary? We have now shown that this same exponential jump in difficulty exists for the $(\exists, k)$-pebble game. It has been conjectured that a similar exponential jump in complexity exists for the k-pebble game. Specifically, the following problem is conjectured to be EXPTIME-complete: Given an integer k, and two structures **A**and **B**, does the Duplicator have a winning strategy for the k-pebble game on **A**and **B**? This conjecture is still open.

In an attempt to better understand the complexity of the k-pebble game, we present a framework to analyze the complexity of a suite of similar combinatorial pebble games. The framework presented here covers some heavily studied pebble games, including Ehrenfeucht-Fraïssé games [25], the $(\exists, k)$-game [23, 22], and the k-pebble game [1, 15], as well as some previously unstudied games.

Each of these games is played on two finite relational structures $\mathbf{A}$ and $\mathbf{B}$, using k pebbles for each of the two players. The first player, Spoiler, takes one of his pebbles and places it on a node in one of the two structures. The second player, Duplicator, responds in the other structure. The Duplicator is attempting to maintain some property of the mapping from pebbles on $\mathbf{A}$ to the pebbles on $\mathbf{B}$. The variations come from how the Spoiler is allowed to choose which structure to play on, what property the Duplicator must maintain, and whether the Spoiler may pick up pebbles in addition to placing them.

With two structures, we must know which player places pebbles on which structure. In a no-alternation game, the Spoiler always places pebbles on A, and the Duplicator always places pebbles on B. This means that the Spoiler has no choice about which structure to play on, and has limited options for plays. We will refer to this as a one-sided game, meaning that the Spoiler can play on only one side. On the other hand, the Spoiler might be allowed to choose which structure to play on, forcing the Duplicator to play on the other one. In this sort of game, the Spoiler may play on either A or B, although the mapping property the Duplicator is trying to maintain still applies to the mapping determined from A to B. Games with this property will

be referred to as two-sided games. One could also consider games with limited alternation, but we will focus here on one-sided and two-sided games.

We consider two basic properties of mappings: Whether the mapping is one-to-one (1-1), and how strictly the mapping must preserve relations. A mapping is 1-1 if no node in B is associated with two distinct nodes in A. To make this more explicit, we pair up the pebbles, such that a_i is the node in A occupied by the ith pebble on A, and b_i is the node in B occupied by the corresponding ith pebble on B. A mapping is 1-1 if for every pair of pebble pairs (a_i, b_i), (a_j, b_j), if $b_i = b_j$ then $a_i = a_j$. A game is known as 1-1 if the Duplicator must maintain the property that the mapping from A to B is always 1-1.

As far as preserving the relations, we consider two kinds of mappings, one-way homomorphism and two-way homomorphism. In a one-way homomorphism, for every relation $R_i^{\mathbf{A}}$, and every tuple $(a_1, \ldots, a_{k_i}) \in R_i^{\mathbf{A}}$, such that $a_1, \ldots, a_{k_i}$ have pebbles on them, the tuple of corresponding pebbled nodes in B, $(b_1, \ldots, b_{k_i}) \in R_i^{\mathbf{B}}$. Notice that one-way homomorphisms are not hurt by extra tuples in $\mathbf{B}$. In the case of undirected graphs, there is always a one-way homomorphism from any graph to a complete graph with the same number of nodes.

In a two-way homomorphism, every tuple in $\mathbf{A}$ must correspond to a tuple in $\mathbf{B}$, and every tuple in $\mathbf{B}$ must correspond to a tuple in $\mathbf{A}$. Specifically, there is a pebbled tuple $(a_1, \ldots, a_{k_i}) \in R_i^{\mathbf{A}}$ if and only if the corresponding pebbled tuple $(b_1, \ldots, b_{k_i}) \in R_i^{\mathbf{B}}$.

The final parameter discussed here is how long the game can last. In Ehrenfeucht-Fraïssé games, each player places pebbles until they have placed them all, and then the game ends. On the other hand, in the $(\exists, k)$-pebble game, the Spoiler can choose to pick up pebbles and place them again, allowing the game potentially to last forever. For the rest of this discussion, we will be considering only infinite play games.

In this framework, the $(\exists, k)$pebble game presented earlier is the one-sided, one-way k-pebble game. The k-pebble game of Immerman and Barwise is the two-sided, 1-1, two-way k-pebble game. For a summary of the complexity results known about games in this framework, see Table 2.1.

2.3.1 One-sided, One-way k-Pebble Game

As discussed earlier, this game corresponds with the $(\exists, k)$-pebble game. As proved above, if k is fixed, then this game is P-complete, and if k is part of the input, then this game is

Game	k **fixed**	k **varied**
1-sided, 1-way game	P-complete - Thm 2.1.6	EXPTIME-complete - Thm 2.2.7
1-sided, 1-1, 1-way game	P-complete - Thm 2.3.1	EXPTIME-complete - Thm 2.3.1
1-sided, 2-way game	P-complete - Thm 2.3.4	EXPTIME-complete - Thm 2.3.4
1-sided, 1-1, 2-way game	P-complete - Thm 2.3.5	EXPTIME-complete - Thm 2.3.5
2-sided, 1-way game	P-complete - Thm 2.3.8	EXPTIME-complete - Thm 2.3.8
2-sided, 1-1, 1-way game	P-complete - Thm 2.3.11	EXPTIME-complete - Thm 2.3.11
2-sided, 2-way game	P-complete - Thm 2.3.14	Unknown
2-sided, 1-1, 2-way game	P-complete [15]	Unknown

Table 2.1: A summary of the complexity results known for pebble games.

EXPTIME-complete.

2.3.2 One-sided, 1-1, One-way k-Pebble Game

In the following theorem, when we say the structures have a *bounded vocabulary*, we mean that there is some fixed p such that no relation in $\mathbf{A}$ or $\mathbf{B}$ has arity greater than p. In addition, for the following theorem we make the assumption that k is no greater than the maximum of $|A|$ and $|B|$. This is a reasonable assumption, because if each node of both $\mathbf{A}$ and $\mathbf{B}$ is covered, then playing extra pebbles does not change the properties of the position. If the Spoiler plays on a node in $\mathbf{A}$, then the Duplicator simply plays according to the mapping that already exists.

Theorem 2.3.1 *On structures over a bounded vocabulary, there is a logspace reduction from the one-sided, one-way game to the one-sided, 1-1, one-way game.*

Proof: Given $\mathbf{A}$, $\mathbf{B}$, and k, construct $\mathbf{A'}$, $\mathbf{B'}$, and k as follows: For each node a_i in A, A' has k copies $a_i^1, \ldots, a_i^k$. For every tuple $(a_1, \ldots, a_q)$ in some relation R of $\mathbf{A}$, the corresponding relation in $\mathbf{A'}$ has $k^{\text{arity}(R)}$ tuples that correspond to all possible combinations of the k version of each node. We construct $\mathbf{B'}$ from $\mathbf{B}$ in a similar way. Note that the number of tuples created is exponential in the arity of the relations. If the maximum arity of the vocabulary were not bounded, this reduction would be exponential in the size of the input.

This reduction is in L, because we simply need a number of counters, equal to the maximum allowed arity of a relation, that can each count up to k, and a single counter that can

count up to the size of A and B. Under the assumption that $k \leq \max(|A|, |B|)$, and that the maximum arity is fixed, this requires a logarithmic amount of space.

Assume that the Spoiler wins the one-sided, one-way game on $\mathbf{A}$, $\mathbf{B}$, k. This means that the Spoiler has a winning strategy to force the Duplicator into a mapping that is not a one-way homomorphism. Any mapping that is not a one-way homomorphism is also not a 1-1, one-way homomorphism, so the same strategy is also a winning strategy for the one-sided, 1-1, one-way game, where any play on a node a_i corresponds to a play on any a_i^j.

Now assume that the Duplicator has a winning strategy for the one-sided, one-way game on $\mathbf{A}$, $\mathbf{B}$, k. To extend this to a winning strategy for the one-sided, 1-1, one-way game on $\mathbf{A}'$, $\mathbf{B}'$, and k, we consider a simulated version of the one-sided, one-way game on $\mathbf{A}$, $\mathbf{B}$, and k. Any time the Spoiler plays on a node a_i^j, the Duplicator first considers how she would play in response to a_i in the one-sided, one-way game. This gives rise to a node b_i. Now the Duplicator need only choose a b_i^k such that the 1-1 property is not violated. Since there are as many such nodes as there are pebbles, the Duplicator always has a valid choice.

□

Corollary 2.3.2 *For each fixed k: Given $\mathbf{A}$ and $\mathbf{B}$, determining whether the Duplicator has a winning strategy for the one-sided, 1-1, one-way game is P-complete.*

Corollary 2.3.3 *Given $\mathbf{A}$, $\mathbf{B}$, and k, determining whether the Duplicator has a winning strategy for the one-sided, 1-1, one-way game is EXPTIME-complete.*

2.3.3 One-sided, Two-way k-Pebble Game

Theorem 2.3.4 *Given $\mathbf{A}$, $\mathbf{B}$, and k, determining whether the Duplicator has a winning strategy for the one-sided, two-way game is EXPTIME-complete.*

Proof: In the proof of the complexity of the one-sided, one-way game, the winning strategy for the Duplicator always maintains a two-way homomorphism. □

2.3.4 One-sided, 1-1, Two-way k-Pebble Game

The following reduction is identical to the reduction for the one-sided, 1-1, one-way k-pebble game.

Theorem 2.3.5 *On structures over a bounded vocabulary, there is a logspace reduction from the one-sided, two-way game to the one-sided, 1-1, two-way game.*

Proof: Given $\mathbf{A}$, $\mathbf{B}$, and k, construct $\mathbf{A'}$, $\mathbf{B'}$, and k as follows: For each node a_i in A, A' has k copies $a_i^1, \ldots, a_i^k$. For every tuple $(a_1, \ldots, a_q)$ in some relation R of $\mathbf{A}$, the corresponding relation in $\mathbf{A'}$ has $k^{\text{arity}(R)}$ tuples that correspond to all possible combinations of the k version of each node. the input. We construct $\mathbf{B'}$ from $\mathbf{B}$ in a similar way.

This reduction is in L, because we simply need a number of counters, equal to the maximum allowed arity of a relation, that can each count up to k, and a single counter that can count up to the size of A and B. Under the assumption that $k \leq \max(|A|, |B|)$, and that the maximum arity is fixed, this requires a logarithmic amount of space.

Assume that the Spoiler wins the one-sided, two-way game on $\mathbf{A}$, $\mathbf{B}$, k. This means that the Spoiler has a winning strategy to force the Duplicator into a mapping that is not a two-way homomorphism. Any mapping that is not a two-way homomorphism is also not a 1-1, two-way homomorphism, so the same strategy is also a winning strategy for the one-sided, 1-1, two-way game, where any play on a node a_i corresponds to a play on any a_i^j.

Now assume that the Duplicator has a winning strategy for the one-sided, two-way game on $\mathbf{A}$, $\mathbf{B}$, k. To extend this to a winning strategy for the one-sided, 1-1, two-way game on $\mathbf{A'}$, $\mathbf{B'}$, and k, we consider a simulated version of the one-sided, two-way game on $\mathbf{A}$, $\mathbf{B}$, and k. Any time the Spoiler plays on a node a_i^j, the Duplicator first considers how she would play in response to a_i in the one-sided, one-way game. This gives rise to a node b_i. Now the Duplicator need only choose a b_i^k such that the 1-1 property is not violated. Since there are as many such nodes as there are pebbles, the Duplicator always has a valid choice.

□

Corollary 2.3.6 *For each fixed k: Given $\mathbf{A}$ and $\mathbf{B}$, determining whether the Duplicator has a winning strategy for the one-sided, 1-1, two-way game is P-complete.*

Corollary 2.3.7 *Given $\mathbf{A}$, $\mathbf{B}$, and k, determining whether the Duplicator has a winning strategy for the one-sided, 1-1, two-way game is EXPTIME-complete.*

2.3.5 Two-sided, One-way k-Pebble Game

Theorem 2.3.8 *On structures over a bounded vocabulary, there is a logspace reduction from the one-sided, one-way game to the two-sided, one-way game.*

Proof: Given $\mathbf{A}$, $\mathbf{B}$, and k an instance of the one-sided, one-way game, we construct $\mathbf{A}'$, $\mathbf{B}$, and k an instance of the two-sided, one-way game. To form A', we add k nodes, $t_1, \ldots t_k$ to A which are not connected to any other nodes in the Gaifman graph. In other words, $t_1, \ldots, t_k$ do not appear in any relations in $\mathbf{A}'$. The Spoiler wins the two-sided, one-way game if at some point in a run of the game there are pebbled nodes $a_1, \ldots, a_j$ in A' such that for some relation symbol R, $(a_1, \ldots, a_j) \in R^A$, but $(h(a_1), \ldots, h(a_j) \notin R^B$. Since the nodes t_i do not participate in any relation, pebbles on them can never contribute to a win for the Spoiler. We now describe the winning strategies for the Spoiler and Duplicator.

Assume that the Spoiler wins the one-sided, one-way game on $\mathbf{A}$, $\mathbf{B}$, and k. Since $\mathbf{A} \subseteq \mathbf{A}'$, and $\mathbf{B}$ has not changed, the Spoiler's winning strategy for the one-sided, one-way game works for the two-sided, two-way game on $\mathbf{A}'$, $\mathbf{B}$, and k.

Now assume that the Duplicator wins the one-sided, one-way game on $\mathbf{A}$, $\mathbf{B}$ and k. If the Spoiler plays on the subset of A' which is isomorphic to A, the Duplicator plays according to his winning strategy for the one-sided, one-way game. If the Spoiler plays on t_i, then the Duplicator can play on any node in B because, as previously discussed, t_i is not in any relation. If the Spoiler plays on any node in B, then the Duplicator plays on some t_i which is not yet pebbled. □

Corollary 2.3.9 *For each fixed k: Given $\mathbf{A}$ and $\mathbf{B}$, determining whether the Duplicator has a winning strategy for the two-sided, one-way game is P-complete.*

Corollary 2.3.10 *Given $\mathbf{A}$, $\mathbf{B}$, and k, determining whether the Duplicator has a winning strategy for the two-sided, one-way game is EXPTIME-complete.*

2.3.6 Two-sided, 1-1, One-way k-Pebble Game

Theorem 2.3.11 *On structures over a bounded vocabulary, there is a logspace reduction from the two-sided, one-way game to the two-sided, 1-1, one-way game.*

Proof: Given $\mathbf{A}$, $\mathbf{B}$, and k, construct $\mathbf{A'}$, $\mathbf{B'}$, and k as follows: For each node a_i in A, $\mathbf{A'}$ has k copies $a_i^1, \ldots, a_i^k$. For every tuple $(a_1, \ldots, a_q)$ in some relation R of $\mathbf{A}$, the corresponding relation in $\mathbf{A'}$ has $k^{\text{arity}(R)}$ tuples that correspond to all possible combinations of the k version of each node. We construct $\mathbf{B'}$ from $\mathbf{B}$ in a similar way.

This reduction is in L, because we simply need a number of counters, equal to the maximum allowed arity of a relation, that can each count up to k, and a single counter that can count up to the size of A and B. Under the assumption that $k \leq \max(|A|, |B|)$, and that the maximum arity is fixed, this requires a logarithmic amount of space.

Assume that the Spoiler wins the two-sided, one-way game on $\mathbf{A}$, $\mathbf{B}$, k. This means that the Spoiler has a winning strategy to force the Duplicator into a mapping that is not a one-way homomorphism. Because any mapping that is not a one-way homomorphism is also not a 1-1, one-way homomorphism, the same strategy is also therefore a winning strategy for the two-sided, 1-1, one-way game, where any play on a node a_i corresponds to a play on any a_i^j.

Now assume that the Duplicator has a winning strategy for the two-sided, one-way game on $\mathbf{A}$, $\mathbf{B}$, k. To extend this to a winning strategy for the two-sided, 1-1, one-way game on $\mathbf{A'}$, $\mathbf{B'}$, and k, we consider a simulated version of the two-sided, one-way game on $\mathbf{A}$, $\mathbf{B}$, and k. Any time the Spoiler plays on a node a_i^j, the Duplicator first considers how she would play in response to a_i in the one-sided, one-way game. This gives rise to a node b_i. Now the Duplicator need only choose a b_i^k such that the 1-1 property is not violated. Since there are as many such nodes as there are pebbles, the Duplicator always has a valid choice.

□

Corollary 2.3.12 *For each fixed k: Given $\mathbf{A}$ and $\mathbf{B}$, determining whether the Duplicator has a winning strategy for the two-sided, 1-1, one-way game is P-complete.*

Corollary 2.3.13 *Given $\mathbf{A}$, $\mathbf{B}$, and k, determining whether the Duplicator has a winning strategy for the two-sided, 1-1, one-way game is EXPTIME-complete.*

2.3.7 Two-sided, Two-way k-Pebble Game

Theorem 2.3.14 *There is a logspace reduction from the two-sided, 1-1, two-way k-pebble game to the two-sided, two-way k pebble game.*

Proof: Given two relational structures $\mathbf{A}$ and $\mathbf{B}$, we construct $\mathbf{A'}$ and $\mathbf{B'}$ by adding to each structure a new binary relation symbol Q, such that $Q^{\mathbf{A'}} = A^2 - \{(x,x)|x \in A\}$ and $Q^{\mathbf{B'}} = B^2 - \{(x,x)|x \in B\}$. Each structure therefore now contains a relation which is a complete graph on its elements. If the Duplicator ever plays so that the position of the game is not a 1-1, two-way homomorphism, then there will be two elements $a_1, a_2 \in A$, such that a_1 and a_2 are mapped to the same element b of B. Since $(a_1, a_2) \in Q^{\mathbf{A'}}$ and $(b,b)\neg \in Q^{\mathbf{B'}}$, the Duplicator loses the game. The winning positions for the Duplicator are exactly the same for both games.

This reduction is in L, because we simply need two counters to create the two complete graphs.

□

2.3.8 Two-sided, 1-1, Two-way k-Pebble Game

This game corresponds to the k-pebble game. Grohe [15] showed that this game is P-complete for every fixed k. The complexity when k is part of the input is still open.

Theorem 2.3.15 *On structures with a bounded vocabulary, there is a logspace reduction from the two-sided, two-way game to the two-sided, 1-1, two-way game.*

Proof: Given $\mathbf{A}$, $\mathbf{B}$, and k, construct $\mathbf{A'}$, $\mathbf{B'}$, and k as follows. For each node a_i in A, A' has k copies $a_i^1, \ldots, a_i^k$. For every tuple $(a_1, \ldots, a_q)$ in some relation R of $\mathbf{A}$, the corresponding relation in $\mathbf{A'}$ has $k^{\text{arity}(R)}$ tuples that correspond to all possible combinations of the k version of each node. We construct $\mathbf{B'}$ from $\mathbf{B}$ in a similar way.

This reduction is in L, because we simply need a number of counters equal to the maximum allowed arity of a relation, that can each count up to k, and a single counter that can count up to the size of A and B. Under the assumption that $k \leq \max(|A|, |B|)$, and that the maximum arity is fixed, this requires a logarithmic amount of space.

Assume that the Spoiler wins the two-sided, two-way game on $\mathbf{A}$, $\mathbf{B}$, k. This means that the Spoiler has a winning strategy to force the Duplicator into a mapping that is not a one-way homomorphism. Since any mapping that is not a two-way homomorphism is also not a 1-1, two-way homomorphism, that strategy is also therefore a winning strategy for the two-sided, 1-1, two-way game, where any play on a node a_i corresponds to a play on any a_i^j.

Now assume that the Duplicator has a winning strategy for the two-sided, two-way game on $\mathbf{A}$, $\mathbf{B}$, k. To extend this to a winning strategy for the two-sided, 1-1, two-way game on $\mathbf{A}'$, $\mathbf{B}'$, and k, we consider a simulated version of the two-sided, two-way game on $\mathbf{A}$, $\mathbf{B}$, and k. Any time the Spoiler plays on a node a_i^j, the Duplicator first considers how she would play in response to a_i in the two-sided, two-way game. This gives rise to a node b_i. Now the Duplicator need only choose a b_i^k such that the 1-1 property is not violated. Since there are as many such nodes as there are pebbles, the Duplicator always has a valid choice.

□

2.4 Concluding Remarks

2.4.1 Summary of Results

The constraint satisfaction problem is, in general, NP-complete. Because this problem has so many applications in such a wide variety of areas, researchers have worked hard to find tractable cases. Many of these "islands of tractability" correspond to expressibility in Datalog. At the root of this connection lie consistency properties and the existential k-pebble game. We have seen that determining the winner of this game is EXPTIME-complete when k is part of the input, and P-complete when k is fixed. The EXPTIME-completeness result implies that establishing strong k-consistency is also EXPTIME-complete when k is part of the input. This result puts limits on how establishing strong k consistency can be used to help solve instances of CSP.

We have also looked at a number of other, related combinatorial games, in an effort to determine the complexity of the k-pebble game. We have used the complexity of the $(\exists, k)$-pebble game to move toward the complexity of the k-pebble game. We presented a framework that parameterizes these pebble games and gives us a spectrum of games between the $(\exists, k)$-pebble game and the k-pebble game in terms of rules. We have also shown that most of the games have the same complexity as the $(\exists, k)$-pebble game, although some results are still open.

2.4.2 Future Work

We would like to know the complexity of the k-pebble game. Based on the results we have seen, we expect the complexity to match the complexity of the other, related games. The games we have studied are parameterized games, in that we have a parameter k which affects the complexity. We have seen that when k is fixed, these pebble games are P-complete, and when k is part of the input, the games are EXPTIME-complete. There is another approach to complexity which makes more explicit the contribution of this parameter. It is called, rationally enough, parameterized complexity (see [8]).

For any conventional complexity class C, we can consider XC, which is the set of all parameterized languages L, such that L_k is in C, for all k. Clearly, all the pebble games we consider are in XP. As Downey and Fellows [8] point out, the results of Kasai, Adachi and Iwata show that their k-pebble game is XP-complete. The reduction presented here does not, however, show that the $(\exists, k)$-pebble game is XP-complete. The reason is that the reduction does not preserve the parameter. The number of pebbles used in the $(\exists, k)$-pebble game directly related, not to the number of pebbles in the KAI game, but to the number of nodes. We would like to know where the $(\exists, k)$-pebble game, and the other games in our framework, lie on the parameterized complexity scale.

Much of the research into CSP has focused on CSP($\mathbf{B}$), where $\mathbf{B}$ is fixed. Because of this, we might want to consider the complexity of determining the winner of the $(\exists, k)$-pebble game when $\mathbf{B}$ is fixed. Consider the structure $\mathbf{B} = (\{b\}, E)$, such that $E = \{(b, b)\}$. There is a homomorphism from every graph to $\mathbf{B}$, so the question of whether the Duplicator has a winning strategy is trivial. On the other hand, we can consider the structure $\mathbf{S} = (\{0, 1\}, R_1, R_2, R_3, R_4)$, where the R_i are the relations from Table 1.1 representing the logical or with different numbers of negations. There is a polynomial time reduction from the 3-SAT problem to the $(\exists, k)$-pebble game with $\mathbf{S}$. The reduction is the same as the reduction from 3-SAT to CSP, except that we use the size of A as value of k. It is not known whether there exists a structure $\mathbf{B}$ such that the $(\exists, k)$-pebble game with $\mathbf{B}$ fixed is EXPTIME-hard.

Part II

Complexity of Data Exchange

Chapter 3

Introduction to Data Exchange

Every relational database has a schema, which determines the organization of the data. In the Data Exchange Problem, the goal is to transform a database from a source schema to a target schema, while satisfying certain given constraints. The source schema, target schema and constraints are together known as a schema mapping. An instance of the Data Exchange Problem is a schema mapping and a source instance. The task is to produce a target instance that satisfies all of the constraints.

In general there can be many solutions. Fagin et al. [9] have proposed what they call a universal solution as the preferred solution. A *universal solution* is a solution that does not contain any extra assumptions. In essence it contains the minimum amount of information necessary to satisfy all the constraints. They also presented an algorithm to compute a universal solution if the constraints are weakly acyclic. Weak acyclicity is a structural limitation on the constraints that effectively puts bounds on the amount of computation needed to generate a universal solution. Their algorithm runs in polynomial time for any fixed schema mapping where the constraints are weakly acyclic.

A necessary step in the process of finding a universal solution is to answer the existence-of-solutions problem: Is there a target instance such that the source instance and target instance satisfy all the constraints? This is an important problem when upgrading database schemas to account for new information that must be stored, or when importing data from an outside source. In the same paper in which they presented the universal solution, Fagin et al. showed that the existence-of-solutions problem is known to be in polynomial time for every fixed schema mapping in which the constraints are weakly acyclic. In [20], we showed that if you relax the weak

acyclicity condition in a minimal way, the problem can be undecidable. In addition we showed that if you allow the schema mapping to vary, but still require the weak acyclicity condition, the problem is in 2EXPTIME. If you put further restrictions on the schema mapping, the the existence-of-solutions problem can be CoNP-Complete or EXPTIME-Complete.

3.1 Background on Data Exchange

A *schema* is a finite collection $\mathbf{R} = (R_1, \ldots, R_k)$ of relation symbols, each of a fixed arity. An *instance* I over $\mathbf{R}$ is a sequence $(R_1^I, \ldots, R_k^I)$ such that each R_i^I is a finite relation of the same arity as R_i. We shall often use R_i to denote both the relation symbol and the relation R_i^I that interprets it. Given a tuple $\mathbf{t}$, we denote by $R(\mathbf{t})$ the association between $\mathbf{t}$ and the relation R where it occurs. Let $\mathbf{S} = (S_1, \ldots, S_n)$ and $\mathbf{T} = (T_1, \ldots, T_m)$ be two disjoint schemas. We call $\mathbf{S}$ the *source* schema and $\mathbf{T}$ the *target* schema. Instances over $\mathbf{S}$ are called *source* instances, and instances over $\mathbf{T}$ are called *target* instances.

We consider schema mappings of the form $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S}$ is a source schema, $\mathbf{T}$ is a target schema, Σ_{st} is a finite set of s-t tgds, and Σ_t is the union of a finite set of target tgds with a finite set of target egds. As described earlier, if I is a source instance, then a *solution for* I is a target instance J such that the pair (I, J) satisfies every constraint in $\Sigma_{st} \cup \Sigma_t$. The *data exchange problem associated with* $\mathcal{M}$ asks, given a source instance I, to construct a solution J for I. The *existence-of-solutions problem for* $\mathcal{M}$ asks: Given a source instance I, does a solution for I exist? Clearly, this decision problem underlies the data exchange problem itself, and any algorithm for the data exchange problem will also solve the existence-of-solutions problem.

As an example, consider the schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ in which $\mathbf{S} = \{E\}$, $\mathbf{T} = \{F\}$, and

$$\begin{aligned} \Sigma_{st} &= \{E(x,z) \rightarrow \exists y(F(x,y) \wedge F(y,z))\} \\ \Sigma_t &= \{F(x,y) \wedge F(y,z) \rightarrow F(x,z), \\ &\qquad F(x,u) \wedge F(x,v) \rightarrow u = v\} \end{aligned}$$

Note that we omitted the universal quantifiers in the above constraints; in the sequel, we will often do this, and implicitly assume such quantification. The source instance $I = \{E(1,2)\}$ has a solution $J = \{F(1,2), F(2,2)\}$. In contrast, no solution exists for the source instance

$I' = \{E(1,2), E(2,1)\}$.

We now give the definition of a weakly acyclic set of target tgds. This crucial concept was formulated by A. Deutsch and L. Popa in 2001, and independently used in [9] and [7] (in the latter paper, under the term *constraints with stratified witness*).

Definition 3.1.1 Let Σ be a set of tgds over a schema $\mathbf{T}$. Construct a directed graph, called the *dependency graph*, as follows:

(i) Nodes: For every pair (R, A) with R a relation symbol of the schema and A an attribute of R, there is a distinct node; call such a pair (R, A) a *position*.

(ii) Edges: For every tgd $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})$ in Σ and for every x in $\mathbf{x}$ that occurs in ψ, and for every occurrence of x in ϕ in position (R, A_i):

1. For every occurrence of x in ψ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$ (if it does not already exist).

2. In addition, for every existentially quantified variable y and for every occurrence of y in ψ in position (T, C_k), add a *special edge* $(R, A_i) \rightarrow (T, C_k)$ (if it does not already exist). Note that there may be two edges in the same direction between two nodes, but exactly one of the two edges is special.

• We say that Σ is *weakly acyclic* if the dependency graph has no cycle going through a special edge.

• We say that a tgd θ is *weakly acyclic* if the singleton set $\{\theta\}$ is weakly acyclic. □

For example, the tgd $E(x,y) \rightarrow \exists z\, E(x,z)$ is weakly acyclic; in contrast, the tgd $E(x,y) \rightarrow \exists z\, E(y,z)$ is not, because the dependency graph contains a special self-loop. Since no existentially quantified variables occur in full tgds, every set of full tgds is weakly acyclic. Note also that if W_1 and W_2 are two weakly acyclic sets of tgds, then their union $W_1 \cup W_2$ need not be weakly acyclic.

Example 3.1.1 *Consider the tgd $R(x,y) \rightarrow \exists w R(x,w)$. As we can see in Figure 3.1a, this tgd is weakly acyclic, because there is no cycle that includes a special edge. Similarly, the tgd $R(x,y) \rightarrow R(y,x)$ in Figure 3.1b is weakly acyclic. Figure 3.1c, however, shows that the two tgds together form a non-weakly acyclic set.*

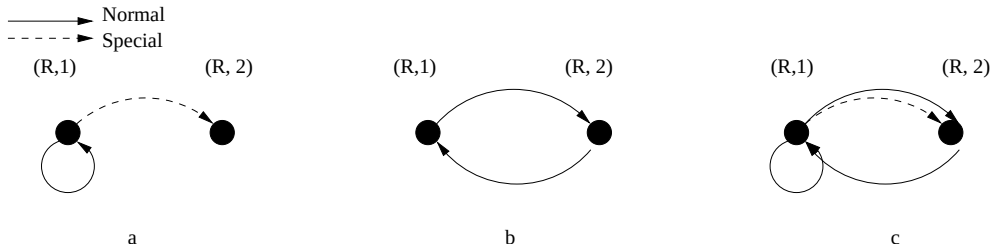

Figure 3.1: a) Dependency graph for $R(x, y) \rightarrow \exists w R(x, w)$. b) Dependency graph for $R(x, y) \rightarrow R(y, x)$. c) Dependency graph for the set containing both dependencies.

In [9], it was shown that weak acyclicity is a sufficient condition for the tractability of the data exchange problem; in particular, it is a sufficient condition for the tractability of the existence-of-solutions problem. Before we state the result, we need a few definitions.

A *homomorphism from a conjunctive formula $\phi(\mathbf{x})$ to an instance K* is similar to a homomorphism between structures. It is a mapping from the variables in $\mathbf{x}$ to the elements of K, such that for each atom in ϕ, the corresponding fact is in K.

Definition 3.1.2 *(Chase Step)*

Let K be an instance. We now define what to do for the two kinds of dependencies we are dealing with, tgds and egds.

Let d be a tgd $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$. Let h be a homomorphism from $\phi(\mathbf{x})$ to K such that there is no extension of h to a homomorphism h' from $\phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y})$ to $\mathbf{K}$. If there is such a homomorphism h, we say that the tgd d can be applied to K with homomorphism h.

Now we extend K based on h and d. First, we extend h to h' by mapping each variable in $\mathbf{y}$ to a fresh labeled null. Then, we extend K to K' by adding the image of the atoms under h'. This is written as $K \xrightarrow{d,h} K'$, which is read as "the result of applying d to K with h is K'."

Let d be an egd $\phi(\mathbf{x}) \rightarrow (x_1 = x_2)$. Let h be a homomorphism from $\phi(\mathbf{x})$ to K such that $h(x_1) \neq h(x_2)$. We say that d can be applied to K with homomorphism h. There are two cases:

- *If both $h(x_1)$ and $h(x_2)$ are constants, then the result of applying d to K with h is "failure", written $K \xrightarrow{d,h} \perp$.*
- *Otherwise, we resolve this by identifying $h(x_1)$ with $h(x_2)$ to form K' in the following way. If one of them is a constant and the other is a labeled null, we replace the null with*

the constant everywhere it appears in K. If they are both labeled nulls, we choose one and replace it in K with the other. Again, this yields $K \xrightarrow{d,h} K'$.

Each transformation of K to K' by $K \xrightarrow{d,h} K'$ constitutes a chase step.

Definition 3.1.3 *Let Σ be a set of egds and tgds, and K be an instance. A chase sequence of K with Σ is simply a sequence of chase steps $K_i \xrightarrow{d_i,h_i} K_{i+1}$, where $i = 0, 1, \ldots$, $K = K_0$, and $d_i \in \Sigma$.*

A finite chase is a finite chase sequence, $K_i \xrightarrow{d_i,h_i} K_{i+1}$, $i = 0, \ldots, m-1$, such that either $K_m = \perp$, or there is no dependency $d*$ and no homomorphism $h*$ such that $d*$ can be applied to K_m with $h*$. If $K_m = \perp$ then the chase is said to be a failing chase. Otherwise we say the chase is successful, and that K_m is the result of the chase.

In general, there may be no finite chase for a given instance and schema mapping. Fagin et al. [9] showed that if there is a finite chase, then the result of a finite chase indicates whether there is a solution or not.

Theorem 3.1.4 *[9] Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a schema mapping where Σ_{st} is a set of tgds and Σ_t is the union of a set of tgds with a set of egds.*

- *Let $\langle I, J\rangle$ be the result of some successful finite chase of $\langle I, \rangle$ with $\Sigma_{st} \cup \Sigma_t$.Then J is a universal solution.*
- *If there exists some failing finite chase of $\langle I, \rangle$ with $\Sigma_{st} \cup \Sigma_t$, then there is no solution.*

This result implies that if we had a condition that guaranteed a finite chase, we would have an algorithm to solve the existence-of-solutions problem. The sufficient condition is weak acyclicity.

Theorem 3.1.5 ([9]) *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is the union of a set of target egds with a weakly acyclic set of target tgds. Then, there is an algorithm that is based on the chase procedure and has the following properties:*

(1) Given a source instance I, the algorithm determines whether a solution for I exists and, if so, it constructs a solution for I. (In fact, it constructs a universal solution.

(2) The running time of the algorithm is bounded by a polynomial in the size of the source instance I.

Note that if $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ is a schema mapping such that Σ_t is a weakly acyclic set of target tgds (in particular, Σ_t contains no target egds), then the existence-of-solutions problem for $\mathcal{M}$ is trivial. This follows from the correctness of the chase procedure. Because Σ_t is a weakly acyclic set of tgds, we know that the chase procedure is finite. A failing chase sequence results from a chase step $K \xrightarrow{d,h} \perp$, where the dependency d is an egd. This means that every chase sequence must be a finite successful chase sequence. It follows that if Σ_t is a set of full target tgds, then the existence-of-solutions problem for $\mathcal{M}$ is trivial.

Chapter 4

Complexity of Data Exchange

Theorem 3.1.5 in the previous chapter is a result about the *data complexity* of data exchange, because the schema mapping $\mathcal{M}$ is assumed to be fixed. In this chapter, we will focus on the *combined complexity* of data exchange, in which the schema mapping is also part of the input.

4.1 Combined Complexity

In this section, we investigate the combined complexity of the existence-of-solutions problem; that is, we study this problem when the input consists of both a schema mapping and a source instance. We begin by formalizing the *existence-of-solutions problem for a class $\mathcal{C}$ of schema mappings* and then proceed to investigate the complexity of this problem.

	Schema Mapping	Existence-of-Solutions Problem
Data Complexity	Fixed, set of target tgds is arbitrary	Can be undecidable
	Fixed, set of target tgds is weakly acyclic	In PTIME; can be PTIME-complete
Combined Complexity	Varies, set of target tgds is weakly acyclic	In 2EXPTIME
	Varies, set of target tgds is weakly acyclic, bounded rank	In EXPTIME; can be EXPTIME-complete
	Schemas are fixed, constraints vary, all tgds are full	In coNP; can be coNP-complete

Table 4.1: Complexity of Data Exchange

Definition 4.1.1 Let $\mathcal{C}$ be a class of schema mappings. The *existence-of-solutions problem for* $\mathcal{C}$ is the following decision problem: Given a schema mapping $\mathcal{M}$ in $\mathcal{C}$, and a source instance I, does a solution for I under $\mathcal{M}$ exists?

It was shown in [20] that if we allow the set of target tgds to be non-weakly acyclic, then the existence-of-solutions problem is potentially undecidable. In view of this, we will only consider schema mappings in which the target tgds form a weakly acyclic set.

We begin by deriving a 2EXPTIME upper bound for the existence-of-solutions problem for a restricted class of schema mappings. To define the class of schema mappings we now need to define an important parameter of the dependency graph. An *incoming path* to a position (R, A) in the dependency graph is a path that ends at (R, A). For each position (R, A) in the graph, define the *rank* of (R, A) to be the maximum number of special edges on any incoming path to (R, A). Because we assume that the dependencies are weakly acyclic, this number is finite for each position. Define the rank of the schema mapping as the maximum rank over all positions in the dependency graph.

Now we consider the class of all schema mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where Σ_{st} is a set of s-t tgds and Σ_t is the union of a set of target egds with a weakly acyclic set of target tgds, such that the rank of $\mathcal{M}$ is bounded by a constant. We prove that in the bounded rank setting, the complexity of the existence-of-solutions problem is in EXPTIME. Observe that the rank of a schema mapping is less than or equal to the number of positions, which means that if the schemas are fixed, the rank is also fixed. Moreover, we identify a family of restricted classes of schema mappings for which this problem is in coNP. We then establish matching EXPTIME-hard and coNP-hard lower bounds. Finally, we prove that slight relaxations of the restrictions cause the complexity of the existence-of-solutions problem to jump from coNP-complete to EXPTIME-complete.

4.2 Combined Complexity: Upper Bounds

It follows from [9] that if we limit ourselves to weakly acyclic sets of tgds, the existence-of-solutions problem is in 2EXPTIME.

Theorem 4.2.1 (Implicit in [9]) *Let $\mathcal{E}$ be the class of all schema mappings $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where Σ_{st} is a set of s-t tgds, and Σ_t is the union of a set of egds with a weakly acyclic set of*

	$\mathbf{S}, \mathbf{T}$	Σ_{st}: tgds	Σ_t: egds and Weakly Acyclic tgds	Combined Complexity	Theorem
Upper Bounds	Vary	Varies	Varies	in 2EXPTIME	4.2.1
	Vary	Varies	Varies; bounded rank	in EXPTIME	4.2.2
	Fixed	Varies; Full tgds	Varies; egds/full tgds	in coNP	4.2.3
	Vary	Varies	Varies; no egds	Trivial	-
Lower Bounds	Vary	Varies; full tgds	Varies; egds/full tgds	EXPTIME-complete	4.3.2
	Fixed	Varies	Varies; egds	EXPTIME-complete	4.3.5
	Fixed	Fixed; full tgds	Varies	EXPTIME-complete	4.3.6
	Fixed	Fixed; Full tgds	Varies; egds	coNP-complete	4.3.7
	Fixed	Varies; full tgds	Fixed; egds	coNP-complete	4.3.9

Table 4.2: Combined Complexity of Data Exchange

tgds. The existence-of-solutions problem for $\mathcal{E}$ is in 2EXPTIME.

Proof: This follows from the proof of Theorem 3.9 of [9]. It was shown in [9] that the length of every chase sequence of I with $\Sigma_{st} \cup \Sigma_t$ is bounded by a polynomial in the size of the instance I. This polynomial is doubly exponential in the size of $\Sigma_{st} \cup \Sigma_t$ and $\mathbf{T}$. We first consider only tgds.

Consider the dependency graph G of $\Sigma = \Sigma_{st} \cup \Sigma_t$. Let p be the number of positions in the dependency graph, D be the number of dependencies, r be the maximum rank of a position, and d be the maximum number of incoming special edges to any node. Note that $r \leq p$. We now put bounds on the maximum number of values that can appear in any instance produced by a chase on Σ. We let n be the number of distinct values that appear in the instance K. Let K' be an instance obtained from K after a chase sequence on Σ. Let $N_0, \ldots, N_r$ be a partition of the positions of G such that each position in N_i has rank i.

We now prove, by induction on i, the following claim:

For every i, there exists a function Q_i with the following properties:

1. *The number of distinct values that occur in K' at positions in N_i is at most $Q_i(n, D, p, d)$.*
2. *$Q_i(n, D, p, d)$ is polynomial in n, D, and p with maximum degree $O(d^i)$*

Base Case: For each position (R, A) in N_0, there are no incoming paths with special edges, so the number of values that can appear is simply the number of values in the original instance n.

Inductive Case: In general, a position can contain values from the original instance, of which there are n; values that are copied from other positions; or values generated by dependencies. Values can only be copied from positions in $N_0, \ldots, N_{i-1}$. If there were an edge from a node in N_j to a node $(R, A) \in N_i$ with $j > i$, then the rank of (R, A) would have to be at least j, which contradicts the fact that $(R, A) \in N_i$. Therefore, the number of values that can be copied into positions in N_i is $P_i(n, D, p, d) = Q_0(n, D, p, d) + Q_1(n, D, p, d) + \cdots + Q_{i-1}(n, D, p, d)$. Clearly if the $Q_j(n, D, p, d)$ meet condition 2, then $P_i(n, D, p, d)$ is a polynomial in the appropriate variables, with a maximum degree of $O(d^{i-1})$.

Now we consider how many values can be generated. Recall that d is the maximum number of special edges entering any position, and the number of values that can be available is $P_i(n, D, p, d)$. For each position, any dependency can generate at most $P_i(n, D, p, d)^d$ values. This gives a maximum of $pP_i(n, D, p, d)^d D$ values generated. Therefore, if we add up the number of initial values, the number of copied values, and the number of generated values, we get $Q_i(n, D, p, d) = n + pP_i(n, D, p, d)^d D + P_i(n, D, p, d)$. By exponentiating $P_i(n, D, p, d)$, we increase the maximum degree from $O(d^{i-1})$ to $O(d^i)$.

By substituting r for i, we now see that $Q_r(n, D, p, d)$ is polynomial in n, D, and p, singly exponential in d, and doubly exponential in r. The maximum number of tuples that can exist in a relation is bounded by $Q_r(n, D, p, d)^p$, because p, being the number of positions, is a bound on the arity of each relation. We multiply $Q_r(n, D, p, d)^p$ by the number of relations s to get a bound on the number of tuples in an instance, $sQ_r(n, D, p, d)^p$. This bound on the number of tuples is still doubly exponential in r, and at worst, exponential in each other variable. Since the chase generates a tuple at each step, the bound on the number of tuples is also a bound on the number of chase steps, showing that this algorithm is in 2EXPTIME.

If we now consider also egds, the upper bound on the number of possible tuples does not change. At each step, we assume, for the sake of the upper bound, that all possible tuples occur. If an egd fires and identifies two values, it doesn't change the maximum number of possible values in the positions of N_i. Because the firing of egds does not change our upper bound on the number of tgds that can fire, we can bound the number of chase steps with egds by noticing that every egd chase step removes one value from the system. This means that there can be no more egd chase steps than the total number of values, $Q_r(n, D, p, d)$.

□

Corollary 4.2.2 *Let r be a fixed integer. Let $\mathcal{E}$ be the class of all schema mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where Σ_{st} is a set of s-t tgds, Σ_t is the union of a set of egds with a weakly acyclic set of tgds, and the rank of $\mathcal{M}$ is bounded by r. The existence-of-solutions problem for $\mathcal{E}$ is in* EXPTIME.

Example 4.2.1 *Here, we present an example of a schema mapping such that any successful chase must take doubly exponential time.*

The schema $\mathbf{S}$ consists of a single unary relation symbol R. The target schema $\mathbf{T}$ has unary relation symbols U_i, U_i', and the ternary relation symbol T_i for $0 <= i <= r$, where r is a parameter. There is only a single tgd in Σ_{st}, which copies the values from R to U_0.

Σ_t*:*

$U_i(x) \rightarrow U_i'(x)$	$0 \leq i \leq r$
$U_i(x) \wedge U_i'(y) \rightarrow \exists N T_i(x, y, N)$	$0 \leq i \leq r$
$T_i(x, y, z) \rightarrow U_{i+1}(x), U_{i+1}(y), U_{i+1}(z)$	$0 \leq i < r$
$T_i(x, y, z) \wedge T_i(x', y', z) \rightarrow x = x'$	$0 \leq i \leq r$
$T_i(x, y, z) \wedge T_i(x', y', z) \rightarrow y = y'$	$0 \leq i \leq r$

Note that the rank of Σ_t is r.

Let n be the number of values in U_0. We show by induction that for each i, U_i must contain at least $n^{(2^i)}$ values.

Base case: $i = 0$. U_0 has n values that are all constants, so U_0 has at least $n^{(2^0)} = n$ values.

Inductive case: Assume that $U_{(i-1)}$ has to contain at least $n^{(2^{(i-1)})}$ values. The tgds force there to be one row in $T_{(i-1)}$ for every possible pair of values from $U_{(i-1)}$. Any attempt to identify nulls to reduce the number of rows in $T_{(i-1)}$ would reduce the number of values in $U_{(i-1)}$ which is a contradiction. This gives us $[n^{(2^{(i-1)})}]^2 = n^{(2^i)}$ rows in $T_{(i-1)}$. Each of these rows must have a distinct value because of the two egds, and the fact that we cannot reduce the number of values incoming from $U_{(i-1)}$. This gives us at least $n^{(2^i)}$ values in the third column of $T_{(i-1)}$. These are copied to U_i.

This gives us at least $n^{(2^r)}$ values in U_r. This means that there are instances for which we cannot compute a solution less than doubly exponential time, because of the size of the instance. This does not, however, prove that we cannot compute the existence-of-solutions problem in EXPTIME.

In contrast, for the classes of schema mappings in which the source and target schemas are fixed, and the s-t tgds and the target tgds vary but are full, the existence-of-solutions problem

is in coNP.

Theorem 4.2.3 *Let $\mathbf{S}^*$ be a fixed source schema and $\mathbf{T}^*$ a fixed target schema. Let $\mathcal{C}(\mathbf{S}^*, \mathbf{T}^*)$ be the class of all schema mappings $(\mathbf{S}^*, \mathbf{T}^*, \Sigma_{st}, \Sigma_t)$ such that Σ_{st} is a set of full s-t tgds and Σ_t is the union of a set of egds with a set of full tgds. The existence-of-solutions problem for $\mathcal{C}(\mathbf{S}^*, \mathbf{T}^*)$ is in* coNP.

Proof: Since $\Sigma_{st} \cup \Sigma_t$ consists of only full tgds, any solution will have size at most $|I|^{|\mathbf{T}|}$, where $|I|$ denotes the number distinct values in I and $|\mathbf{T}|$ denotes the sum of arities of each relation symbol in $\mathbf{T}$. Since $\mathbf{T}$ is fixed, it is easy to see that the size of any solution is polynomial in the size of I.

If there is no solution, it must be that some egd in Σ_t is violated. We can therefore guess a failing chase sequence, where the last chase step is an application of the egd that causes the violation. Since Σ_{st} and Σ_t are full tgds, only the last chase step is an egd chase step; every other chase step in this sequence is a tgd chase step. Because polynomially many tuples can be generated in the target, and each tgd chase step generates at least one new tuple in the target instance, we conclude that the length of this failing chase sequence is polynomial in the size of I.

Clearly, verifying that this is a failing chase sequence can be done in polynomial time: (1) for every chase step that generates K_2 from K_1 with tgd σ and homomorphism h, let σ be of the form $\phi(\mathbf{x}) \rightarrow \psi(\mathbf{x})$. We verify that every tuple in $\phi(h(\mathbf{x}))$ is from K_1, that some tuple in $\psi(h(\mathbf{x}))$ does not exist in K_1, and that K_2 is the union of K_1 with the tuples $\psi(h(\mathbf{x}))$. This verification takes polynomial time. (2) For the last step, where an egd of the form $\phi'(\mathbf{x}) \rightarrow x_1 = x_2$ is applied on the instance K_1, we verify that every tuple in $\phi'(h(\mathbf{x}))$ is from K_1 and that $h(x_1) \neq h(x_2)$. Clearly, this verification step also takes polynomial time. □

It is worth noting that if Σ_t consists of only a weakly acyclic set of tgds (and no egds), then every source instance has a solution and so the existence-of-solutions problem is trivial. Indeed, in this case, the algorithm of [9] for computing a solution will always terminate and produce a solution, because no finite chase is failing. This observation and the preceding Theorems 4.2.2 and 4.2.3 are summarized in the first three rows of Table 4.2.

We now proceed to derive EXPTIME-hard and coNP-hard lower bounds; the presence of target egds will play a crucial role in proving these results.

4.3 Combined Complexity: Lower Bounds

In this section, we show that the existence-of-solutions problem for the class $\mathcal{E}$ of schema mappings in Corollary 4.2.2 is EXPTIME-complete. We also show that there are fixed schema mappings $\mathbf{S}^*$ and $\mathbf{T}^*$ such that the existence-of-solutions problem for the class $\mathcal{C}(\mathbf{S}^*, \mathbf{T}^*)$ in Theorem 4.2.3 is coNP-complete. Moreover, we establish that if we consider slight extensions of the classes $\mathcal{C}(\mathbf{S}^*, \mathbf{T}^*)$, then the existence-of-solutions problem may become EXPTIME-complete. Specifically, this problem may become EXPTIME-complete once we consider at least one of the following extensions of $C(\mathbf{S}^*, \mathbf{T}^*)$:

(1) Allow the schemas to vary (Theorem 4.3.2).

(2) Allow the s-t tgds to be non-full (Theorem 4.3.5).

(3) Allow the target tgds to be non-full (Theorem 4.3.6).

Actually, Theorem 4.3.5 holds even when all target dependencies are egds, and Theorem 4.3.6 holds even when the s-t tgds are fixed and full. These results are summarized in the first three rows of the lower part of Table 4.2.

We also show that the existence-of-solutions problem may be coNP-complete even for a class of schema mappings where the schemas are fixed, the s-t tgds are full and held fixed, and the target dependencies are allowed to vary, but all are egds (Theorem 4.3.7). If the target egds are also fixed, then, as we have shown earlier, the (data) complexity of the existence-of-solutions problem for a fixed schema mapping is PTIME-complete. Hence, a slight extension to a fixed schema mapping, by allowing the target egds to vary, may cause the complexity to become coNP-complete. We also show that the existence-of-solutions problem for another minimal extension may be coNP-complete; now the target egds are fixed, but we allow full s-t tgds that can vary (Theorem 4.3.9). These results are summarized in the last two rows of Table 4.2.

4.3.1 EXPTIME-Completeness

Our EXPTIME-hardness reductions make use of a combined complexity result of *Datalog sirups* from Gottlob and Papadimitriou [14], which we describe next.

Single-rule Datalog programs of [14] A *single-rule program (sirup)* is a Datalog program with one rule and a number of *initializations* consisting of facts. We briefly review the Datalog

terms defined in Part 1. The rule is of the form: $A_0 \leftarrow A_1, ..., A_m$, where each A_i, $0 \leq i \leq m$, is an atom. An *atom* is a formula $p(t_1, ..., t_n)$, where p is a predicate symbol of arity n and t_i, $1 \leq i \leq n$, is a variable or a constant. The *head* of the rule is A_0, and $A_1, ..., A_m$ is referred to as the *body* of the rule. The predicate symbol that appears in A_0 is called an *intensional database predicate (idb)* symbol; those that appear only in the body of the rule are called *extensional database predicate (edb)* symbols. A rule of the form $A_0 \leftarrow$, (with an empty body) is a *fact*. We say the fact is *ground* if every term that appears in A_0 is a constant. The edb symbols occur in a database $\mathbf{D}$, and if the idb symbol also occurs in the body of a sirup rule, it is initialized by facts. A *single ground fact (SGF) sirup* is a Datalog program with one rule and at most one ground fact. Gottlob and Papadimitriou [14] investigated the combined complexity of the following decision problem: Given a Datalog program P, a database $\mathbf{D}$, and a ground fact δ, is it the case that $P \cup \mathbf{D} \models \delta$? In other words, is δ derivable from $\mathbf{D}$ via P? They showed that even if Datalog programs are limited to SGF sirups, the combined complexity is EXPTIME-complete. We refer to this decision problem as the *SGF sirup problem*.

Theorem 4.3.1 (Combined complexity of SGF sirups [14]) *The combined complexity of the SGF sirup problem is* EXPTIME*-complete.*

Using Theorem 4.1 of [14], we show that the existence-of-solutions problem for $\mathcal{C}$ is EXPTIME-hard, where $\mathcal{C}$ is the class of schema mappings where we allow only full s-t tgds and Σ_t is the union of a set of egds with a set full tgds. Note that this is only a slight extension of the class $\mathcal{C}(\mathbf{S}^*, \mathbf{T}^*)$ of schema mappings in Theorem 4.2.3; here, the source and target schemas are allowed to vary. Note that in this case the maximum rank of a position is 0, because there are no special edges in the dependency graph.

Theorem 4.3.2 *Let $\mathcal{C}$ be the class of all schema mappings $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where Σ_{st} is a set of full s-t tgds and Σ_t is the union of a set of egds with a set of full tgds. The existence-of-solutions problem for $\mathcal{C}$ is* EXPTIME*-hard.*

Proof: We are given an SGF sirup P, a database $\mathbf{D}$, and a fact δ; we wish to determine whether $P \cup \mathbf{D} \models \delta$. Let the sirup rule be of the form $A(\mathbf{x}) \leftarrow \mathbf{Q}_1(\mathbf{x_1}), \ldots, \mathbf{Q}_n(\mathbf{x_n})$, where each symbol $\mathbf{Q}_i$, $1 \leq i \leq n$, represents either an extensional database predicate or the intensional database predicate A. If A occurs among $\mathbf{Q}_1, ..., \mathbf{Q}_n$, then A is initialized by a ground fact. Let the arity of A be k and let δ denote the fact $A(c_1, ..., c_k)$.

Based on P, $\mathbf{D}$, δ, and the ground fact, we construct a schema mapping and a source instance I as follows: The source schema $\mathbf{S}$ consists of all the extensional database predicates $R_1, ..., R_m$, as well as two relational symbols A and W. Each R_i, $1 \leq i \leq m$, has the same arity as the corresponding relation in $\mathbf{D}$, A has arity k, and W has arity $k+1$. The target schema is isomorphic to $\mathbf{S}$. It consists of the relational symbols $R'_1, \ldots, R'_m$, A', and W'. For Σ_{st}, we create $m + 2$ full s-t tgds to copy each source relation to its corresponding target relation:

$$\begin{aligned}\Sigma_{st}: \quad & R_i(\mathbf{x}) \rightarrow R'_i(\mathbf{x}), 1 \leq i \leq m \\ & A(\mathbf{x}) \rightarrow A'(\mathbf{x}) \\ & W(\mathbf{x}) \rightarrow W'(\mathbf{x})\end{aligned}$$

Next, we construct two dependencies in Σ_t. The first is a full tgd that "implements" the sirup rule. The second is an egd that tests the existence of δ.

$$\begin{aligned}\Sigma_t: \quad & \mathbf{Q}'_1(\mathbf{x}_1) \wedge \cdots \wedge \mathbf{Q}'_n(\mathbf{x}_n) \rightarrow A'(\mathbf{x}) \\ & A'(x_1, x_2, \ldots, x_k) \wedge W'(x_1, x_2, \ldots, x_k, y) \rightarrow x_1 = y\end{aligned}$$

In the first tgd, $\mathbf{Q}_i$ represents the ith predicate symbol, $1 \leq i \leq m$, in the body of the sirup rule P. For example, if the sirup rule is $A(x_1) \leftarrow R_2(x_1, x_2), R_1(x_2, x_3), A(x_3)$, then the first tgd in Σ_t is $R'_2(x_1, x_2) \wedge R'_1(x_2, x_3) \wedge A'(x_3) \rightarrow A'(x_1)$.

We create a source instance I as follows: First, we populate $R_1, \ldots, R_m$, and A with the tuples from the corresponding relations in $\mathbf{D}$ and the ground fact, respectively. Then, we populate W with a single tuple $(c_1, c_2, \ldots, c_k, d)$, where d is a fresh symbol that does not occur elsewhere in I.

Clearly, the schema mapping and source instance I can be constructed with one pass through the sirup input P, $\mathbf{D}$, δ, and the ground fact. We show next that there is no solution for I under the constructed schema mapping if and only if $P \cup \mathbf{D} \models \delta$. If there is no solution, then there is a failing chase sequence that applies the egd in Σ_t. Since any application of the egd in a chase sequence must make use of the tuple $W'(c_1, ..., c_k, d)$, it must be that $A'(c_1, ..., c_k)$ also exists in the target instance. Thus, we conclude that δ is derivable from $\mathbf{D}$ via P. Conversely, if $P \cup \mathbf{D} \models \delta$, then chasing I with $\Sigma_{st} \cup \Sigma_t$ will produce $A'(c_1, ..., c_k)$. Since $W'(c_1, ..., c_k, d)$ exists in the target instance, the egd can be applied, and therefore the chase fails. This means that there is no solution. □

Corollary 4.3.3 *Fix an integer r. Let $\mathcal{C}$ be the class of all schema mappings $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where Σ_{st} is a set of s-t tgds and Σ_t is the union of a set of egds with a weakly acyclic set of tgds, such that the maximum rank of any position is at most r. The existence-of-solutions problem for $\mathcal{C}$ is* EXPTIME*-complete.*

The above corollary is a consequence of Corollary 4.2.2 and Theorem 4.3.2. Observe that in the reduction of Theorem 4.3.2, a target tgd was used to implement the sirup rule. In Theorem 4.3.5, we show that even when the schemas are kept fixed and there are no target tgds, but we allow non-full s-t tgds, then the EXPTIME-hard lower bound continues to hold. Toward Theorem 4.3.5, we first prove a lemma. The proof of this lemma shows how we can avoid using a target tgd to implement the sirup rule.

Lemma 4.3.4 *Let $\mathcal{C}$ be the class of all schema mappings $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where Σ_t consists of only egds. The existence-of-solutions problem for $\mathcal{C}$ is* EXPTIME*-hard.*

Proof: This proof is by reduction from the SGF sirup problem. The basic setup is similar to the proof of Theorem 4.3.2. The difference is that since we only have egds in Σ_t (no tgds are now allowed in Σ_t), we employ a different method to track which tuples are in the result of the sirup. Note that since there are no tgds in Σ_t, the maximum rank of any position is 1.

We are given an SGF sirup rule P, a database $\mathbf{D}$, and a fact δ; we wish to determine whether $P \cup \mathbf{D} \models \delta$. Let the sirup rule be of the form $A(\mathbf{x}) \leftarrow \mathbf{Q}_1(\mathbf{x_1}), \ldots, \mathbf{Q}_n(\mathbf{x_n})$, where each symbol $\mathbf{Q}_i$, $1 \leq i \leq n$, represents either an extensional database predicate or the intensional database predicate A. If A occurs among $\mathbf{Q}_1, ..., \mathbf{Q}_n$, then we assume that there is a ground fact that initializes A. Let the arity of A be k and let δ denote the fact $A(c_1, ..., c_k)$.

Based on P, $\mathbf{D}$, δ, and the ground fact, we construct a schema mapping and a source instance I as follows: The source schema $\mathbf{S}$ consists of all the extensional database predicates $R_1, ..., R_m$, as well as four relational symbols A, W, U, and V. Note that the relational symbol A in $\mathbf{S}$ is the same as the intensional database predicate of the sirup P. Each R_i, $1 \leq i \leq m$, has the same arity as the corresponding relation in $\mathbf{D}$, and A has arity k, W has arity $k+1$, and the relations U and V are unary. The target schema consists of the relational symbols $R'_1, \ldots, R'_m$, A', W', U', and V'. The arity of each relation in $\mathbf{T}$ is the same as the corresponding relation in $\mathbf{S}$ except for A' which has arity $k+1$. For Σ_{st}, we create the following s-t tgds:

$$
\begin{aligned}
\Sigma_{st}: \quad & R_1(\mathbf{x}) \rightarrow R'_1(\mathbf{x}) \\
& \ldots \\
& R_m(\mathbf{x}) \rightarrow R'_m(\mathbf{x}) \\
& U(x) \rightarrow U'(x) \\
& W(x_1, \ldots, x_k, x) \rightarrow W'(x_1, \ldots, x_k, x) \\
& U(x) \wedge A(x_1, \ldots, x_k) \rightarrow A'(x_1, \ldots, x_k, x) \\
& V(x_1) \wedge \cdots \wedge V(x_k) \rightarrow \exists N\, A'(x_1, \ldots, x_k, N)
\end{aligned}
$$

The dependencies in Σ_t consist of the following egds:

$$\Sigma_t: \quad U'(y) \wedge A'(x_1, \ldots, x_k, x) \wedge \mathbf{Q}'_1(\mathbf{y}_1) \wedge \cdots \wedge \mathbf{Q}'_n(\mathbf{y}_n) \rightarrow x = y$$
$$U'(y) \wedge W'(x_1, \ldots, x_k, z) \wedge A'(x_1, \ldots, x_k, y) \rightarrow y = z$$

The symbol $\mathbf{Q}'_i$, $1 \leq i \leq n$, represents the underlying predicate symbols. We create the source instance I as follows. As in the proof of Theorem 4.3.2, we populate $R_1, \ldots, R_n$, and A with tuples from the corresponding relations in $\mathbf{D}$ and the ground fact. For W, we create a single tuple $(c_1, c_2, \ldots, c_k, d)$, where d is a fresh value that does not occur elsewhere in I. Finally, for U we create a single tuple with the value "1", and for V we insert all distinct values of P, $\mathbf{D}$, δ, and the ground fact, each as a unary tuple in V.

The first m tgds of Σ_{st} copy the R_i to R'_i, where $1 \leq i \leq m$. The next two s-t tgds copy tuples in U and W to U' and W', respectively. The last two s-t tgds in Σ_{st} build the relation A': One tgd copies every tuple from A to A' along with an additional column which contains the value "1". The last tgd generates all possible tuples composed of values from the source instance I, attaches a labeled null, and puts the resulting tuple into the target relation A'. Intuitively, a labeled null indicates that the tuple is not in the result of applying the sirup rule, and a value "1" indicates otherwise. The set Σ_t is constructed in such a way that the last column v of a tuple $(a_1, ..., a_k, v)$ in A' will be set to "1" if and only if $P \cup \mathbf{D} \models A(a_1, ..., a_k)$. The first egd in Σ_t simulates the sirup rule. In particular, it sets the last column v of a tuple $(a_1, ..., a_k, v)$ to "1" whenever $(a_1, ..., a_k)$ is generated by the sirup rule. For example, if the sirup rule is $A(x_1) \leftarrow R_2(x_1, x_2), R_1(x_2, x_3), A(x_3)$, then the first egd in Σ_t is $U'(y) \wedge A'(x_1, x) \wedge R'_2(x_1, x_2) \wedge R'_1(x_2, x_3) \wedge A'(x_3, y) \rightarrow x = y$. The second egd is similar to the functionality of the egd in the proof of Theorem 4.3.2. It detects whether the fact δ is in the result of the sirup rule. If so, this egd will be violated because it equates "d" with "1".

It is easy to see that the schema mapping and the source instance I can be constructed in polynomial time. Moreover, there is no solution to the schema mapping with source instance I if and only if $P \cup \mathbf{D} \models \delta$. If there is no solution, we know that the chase must fail on one of the egds in Σ_t. Since the chase of I with Σ_{st} only adds the tuple "1" to U' and no other tuples are added to U', we know that y is always bound to the value "1" with the first egd in Σ_t. Since the last column of every tuple in A' is either a labeled null or the value "1", this egd cannot cause a violation. We therefore conclude that the violation is due to the second egd in Σ_t. Since the chase only adds the tuple $(c_1, ..., c_k, d)$ to W' and y always binds to the value "1", the violation must have been due to the existence of a tuple $A'(c_1, ..., c_k, 1)$. Therefore, we conclude that $P \cup \mathbf{D} \models \delta$. It is easy to see that if $P \cup \mathbf{D} \models f$, then there will be no solution,

since the last egd of Σ_t can never be satisfied. □

Using Lemma 4.3.4, we are now ready to show that the EXPTIME-hard lower bound continues to hold even when we fix the source and target schemas.

Theorem 4.3.5 *There exists a fixed source schema $\mathbf{S}^*$ and a fixed target schema $\mathbf{T}^*$ such that the existence-of-solutions problem for $\mathcal{C}^*$ is* EXPTIME*-hard, where $\mathcal{C}^*$ is the class of all schema mappings $(\mathbf{S}^*, \mathbf{T}^*, \Sigma_{st}, \Sigma_t)$ such that Σ_{st} consists of s-t tgds with at most two existentially-quantified variables and Σ_t consists of only two egds.*

Proof: We prove this by a reduction from the SGF sirup problem. We are given an SGF sirup P, a database $\mathbf{D}$, and a fact δ; we wish to determine whether $P \cup \mathbf{D} \models \delta$. Let the sirup rule be of the form: $A(\mathbf{x}) \leftarrow \mathbf{Q}_1(\mathbf{x_1}), \ldots, \mathbf{Q}_n(\mathbf{x_n})$, where each symbol $\mathbf{Q}_i$, $1 \leq i \leq n$, represents either an extensional database predicate or the intensional database predicate A. If A occurs among $\mathbf{Q}_1, ..., \mathbf{Q}_n$, then there is a ground fact that initializes A. Let the arity of A be k and let δ denote the fact $A(c_1, ..., c_k)$.

The source schema $\mathbf{S}$ contains the relational symbols O, G, W, A, U, and V, where O is a binary relation that stores a linear order of the relations involved, G is a relation symbol of arity 5 that stores all tuples of each relation, W is a ternary relation that stores the fact δ, and A is a ternary relation that stores the initialization tuples. Note that the relational symbol A in $\mathbf{S}$ is the same as the intensional database predicate of the sirup P. The relations U and V are unary, where U stores the single value "1" and V stores all possible values that occur in $P \cup \mathbf{D} \models \delta$. The target schema $\mathbf{T}$ is an isomorphic copy of $\mathbf{S}$ and consists of the relational symbols O', G', W', A', U', and V'.

Next, we show how the source instance I is constructed based on P, $\mathbf{D}$, δ, and the ground fact before we describe the construction of Σ_{st} and Σ_t. Suppose $R_1, \ldots, R_m$ are the extensional database predicates, and A is the only intensional database predicate. We now construct O to contain a linear order on the relational symbols that occur in $\mathbf{D}$. That is, we define $O = \{(R_1, R_2), ..., (R_{m-1}, R_m)\} \cup \{(R_m, A)\}$, Next, we construct the relation W. We use W to hold the test tuple δ by defining $W = \{(1, 2, c_1), (2, 3, c_2), \ldots, (k, k+1, c_k)\} \cup \{(k+1, k+2, d)\}$. Intuitively, the ordering of the values in δ is maintained by the pair of numbers in the first two columns of each tuple. The value d is not used anywhere else and is stored as an extra column to the tuple δ. We will subsequently use the existence of d to test if a solution was found.

Finally, we construct the relation G. For every tuple $(a_1, \ldots, a_{k_i})$ of a relation R_i, where R_i has arity k_i, we create k_i tuples in G: $(R_i, 1, 2, n, a_1), (R_i, 2, 3, n, a_2), \ldots, (R_i, k_i, k_i + 1, n, a_{k_i})$. The value in the first column of a tuple is used to distinguish among tuples from different relations. The fourth column holds a value n, which is a unique identifier for the tuple; every tuple has a different identifier. Since the tuple is now split among k_i tuples, the second and third column of the tuple n is used to encode the ordering of values in the original tuple. We repeat this process for the ground fact, if it exists. We create a single tuple "1" in U, and for V we insert all distinct values of P, $\mathbf{D}$, δ, and the ground fact, each as a unary tuple in V.

We describe next the construction of Σ_{st} and Σ_t with an example first. Suppose the sirup rule P is $A(x_1) \leftarrow R_2(x_1, x_2) \wedge R_1(x_2, x_3) \wedge A(x_3)$. Then, the set Σ_{st} consists of the following s-t tgds:

1) $G(u, v, x, y, z) \rightarrow G'(u, v, x, y, z)$
2) $U(x) \rightarrow U'(x)$
3) $W(x_1, x_2, y) \rightarrow W'(x_1, x_2, y)$
4) $O(x, y) \rightarrow O'(x, y)$
5) $U(y) \wedge O(u_1, u_2) \wedge \cdots \wedge O(u_{m-1}, u_m) \wedge O(u_m, u_0) \wedge$
   $G(u_0, s_1, s_2, y_0, x_1) \wedge \cdots \wedge G(u_0, s_k, s_{k+1}, y_0, x_k)$
   $\rightarrow G'(u_0, s_1, s_2, y_0, x_1) \wedge \cdots \wedge G'(u_0, s_k, s_{k+1}, y_0, x_k) \wedge$
   $G'(u_0, s_{k+1}, s_{k+2}, y_0, y)$
6) $V(x_1) \wedge \cdots \wedge V(x_k) \wedge$
   $O(u_1, u_2) \wedge \cdots \wedge O(u_{m-1}, u_m) \wedge O(u_m, u_0)$
   $\rightarrow \exists N \exists M (G'(u_0, s_1, s_2, N, x_1) \wedge \cdots \wedge$
   $G'(u_0, s_k, s_{k+1}, N, x_k) \wedge G'(u_0, s_{k+1}, s_{k+2}, N, M))$

The first four s-t tgds copy the relations G, U, W, and O to G', U', W', and O', respectively. The fifth and sixth s-t tgds build the relation A in G'. The fifth s-t tgd copies every tuple in G that belongs to A over to G' and extends the tuple with a value "1" at the $(k+1)$th column. The sixth s-t tgd generates every possible output tuple of the sirup rule along with a labeled null N that is the identifier of the output tuple. An additional labeled null M is also generated at the $(k+1)$th column; it will be used to indicate whether this tuple exists in the result of the sirup rule. The dependencies in Σ_t consists of the following two egds:

1) $U'(y) \wedge O'(u_1, u_2) \wedge \cdots \wedge O'(u_{m-1}, u_m) \wedge O'(u_m, u_0) \wedge$
$G'(u_0, s_1, s_2, y_0, x_1) \wedge \cdots \wedge G'(u_0, s_k, s_{k+1}, y_0, x_k) \wedge$
$G'(u_0, s_{k+1}, s_{k+2}, y_0, z) \wedge$
$G'(\mathbf{q_1}, s_1^1, s_2^1, y_1, x_1^1) \wedge \cdots \wedge G'(\mathbf{q_1}, s_k, s_{k+1}, y_1, x_{k_1}^1)$
...
$G'(\mathbf{q_n}, s_1^n, s_2^n, y_n, x_1^n) \wedge \cdots \wedge G'(\mathbf{q_n}, s_k, s_{k+1}, y_n, x_{k_1}^n)$
$\rightarrow z = y$

2) $U'(y) \wedge O'(u_1, u_2) \wedge \cdots \wedge O'(u_{m-1}, u_m) \wedge O'(u_m, u_0) \wedge$
$W'(s_1, s_2, x_1) \wedge \cdots \wedge W'(s_k, s_{k+1}, x_k) \wedge$
$W'(s_{k+1}, s_{k+2}, z) \wedge$
$G'(u_0, p_1, p_2, y_0, x_1) \wedge \cdots \wedge G'(u_0, p_k, p_{k+1}, y_0, x_k) \wedge$
$G'(u_0, p_{k+1}, p_{k+2}, y_0, y)$
$\rightarrow y = z$

The first egd is a rewriting of the first tgd in Σ_t of the proof of Lemma 4.3.4 based on the fixed schema $\mathbf{T}$. It simulates the sirup rule. In particular, it sets the last column of a tuple in G' to "1" whenever the tuple is generated by the sirup rule. The second and third line above represent the original relational atom $A'(x_1, ..., x_k, z)$, the fourth line represents the original relational atom $\mathbf{Q}_1(\mathbf{x}_1)$, and so on. Similarly, the W' relational atoms above represent the original relational atom $W'(x_1, ..., x_k, z)$ and the second and third line of the last egd above represent the original relational atom $A'(x_1, ..., x_k, y)$. For example, if the sirup rule is $A(x_1) \leftarrow R_2(x_1, x_2), R_1(x_2, x_3), A(x_3)$, then the first egd in Σ_t is

$U'(y) \wedge O(u_1, u_2) \wedge O(u_2, u_0) \wedge$
$G'(u_0, s_1, s_2, y_0, x_1) \wedge G'(u_0, s_2, s_3, y_0, z) \wedge$
$G'(u_2, p_1, p_2, y_1, x_1) \wedge G'(u_2, p_2, p_3, y_1, x_2) \wedge$
$G'(u_1, q_1, q_2, y_2, x_2) \wedge G'(u_1, q_2, q_3, y_2, x_3) \wedge$
$G'(u_0, r_1, r_2, y_3, x_3) \rightarrow y = z$

and the second egd is

$U'(y) \wedge O(u_1, u_2) \wedge O(u_2, u_0) \wedge$
$W'(v_1, v_2, x) \wedge W'(v_2, v_3, z)$
$G'(u_0, r_1, r_2, y_0, x) \wedge G'(u_0, r_2, r_3, y_0, y) \rightarrow y = z$

It is easy to see that an argument similar to that in the proof of Lemma 4.3.4 shows that there is no solution to the schema mapping with source instance I if and only if $P \cup \mathbf{D} \models \delta$. □

The last result of this section shows that a slight extension of the classes $\mathcal{C}(\mathbf{S}^*, \mathbf{T}^*)$ of schema mappings with a non-full but weakly acyclic target tgd may also cause the complexity of the existence-of-solutions problem to become EXPTIME-hard. In fact, this EXPTIME-hard lower bound holds even when the s-t tgds are held fixed.

Theorem 4.3.6 *There exist a fixed source schema $\mathbf{S}^*$, a fixed target schema $\mathbf{T}^*$, and a fixed set Σ_{st}^* such that*

(1) *Σ_{st}^* consists of full s-t tgds.*

(2) *The existence-of-solutions problem for $\mathcal{C}^*$ is* EXPTIME*-hard, where $\mathcal{C}^*$ is the class of all schema mappings $(\mathbf{S}^*, \mathbf{T}^*, \Sigma_{st}^*, \Sigma_t)$ in which Σ_t consists of one egd and one weakly acyclic tgd with one existentially-quantified variable.*

Proof: We prove this by a reduction from the SGF sirup problem. We are given a SGF sirup P, a database $\mathbf{D}$, and a fact δ; we wish to determine whether $P \cup \mathbf{D} \models \delta$. Let the sirup rule be of the form $A(\mathbf{x}) \leftarrow \mathbf{Q}_1(\mathbf{x_1}), \ldots, \mathbf{Q}_n(\mathbf{x_n})$, where each symbol $\mathbf{Q}_i$, $1 \leq i \leq n$, represents either an extensional database predicate or the intensional database predicate A. If A occurs among $\mathbf{Q}_1, ..., \mathbf{Q}_n$, then we assume that there is a ground fact that initializes A. Let the arity of A be k and let δ denote the fact $A(c_1, ..., c_k)$.

The source schema $\mathbf{S}$ consists of three relational symbols O, W, and G, where O is a binary relation symbol that holds a linear order on the relation symbols that occur in $\mathbf{D}$, W is a ternary relational symbol that is used to hold the fact δ to be tested, and G is a relation symbol of arity 5 that is used to hold all tuples in $\mathbf{D}$. The target schema $\mathbf{T}$ is an isomorphic copy of $\mathbf{S}$ with the relational symbols O', W', and G'.

The s-t tgds in Σ_{st} are fixed, and they simply copy each source relation to its corresponding relation in the target. Hence, Σ_{st} consists of three full s-t tgds:

$$\begin{aligned} \Sigma_{st}: \quad & O(x,y) \rightarrow O(x,y) \\ & W(x,y,z) \rightarrow W'(x,y,z) \\ & G(u,v,x,y,z) \rightarrow G'(u,v,x,y,z) \end{aligned}$$

Next, we show how the source instance I is constructed based on P, $\mathbf{D}$, δ, and the ground fact before we describe the construction of Σ_t. Let $R_1, \ldots, R_m$ be the extensional database predicates, and A be the intensional database predicate. We now construct O to contain a linear order on the relational symbols. That is, we define $O = \{(R_1, R_2), ..., (R_{m-1}, R_m)\} \cup \{(R_m, A)\}$. Next, we construct the relation W. We use W to hold the test tuple δ by defining $W = \{(1, 2, c_1), (2, 3, c_2), \ldots, (k, k+1, c_k)\} \cup \{(k+1, k+2, d)\}$. Intuitively, the ordering of the values in δ is maintained by the pair of numbers in the first two columns of each tuple. The value d is not used anywhere else and is stored as an extra column to the tuple δ. We will subsequently use the existence of d to test if a solution was found. Finally, we construct the relation G. For every tuple $(a_1, \ldots, a_{k_i})$ of every relation R_i, where R_i has arity k_i, we create

k_i tuples in G: $(R_i, 1, 2, n, a_1), (R_i, 2, 3, n, a_2), \ldots, (R_i, k_i, k_i + 1, n, a_{k_i})$. The value in the first column of a tuple is used to distinguish among tuples from different relations. The fourth column holds a value n, which is a unique identifier for the tuple; every tuple has a different identifier. Since the tuple is now split among k_i tuples, the second and third columns of the tuple n are used to encode the ordering of values in the original tuple. We repeat this process for the ground fact, if it exists.

Now we describe the construction of Σ_t. Before we present the encoding, it is instructive to first look at an example. As described earlier, suppose our sirup rule is $A(x_1) \leftarrow R_2(x_1, x_2) \wedge R_1(x_2, x_3) \wedge A(x_3)$; then Σ_t contains the tgd:

$$O'(u_1, u_2) \wedge O'(u_2, u_3) \wedge$$
$$G'(u_2, p_1, p_2, y_2, x_1) \wedge G'(u_2, p_2, p_3, y_2, x_2) \wedge$$
$$G'(u_1, s_1, s_2, y_1, x_2) \wedge G'(u_1, s_2, s_3, y_1, x_3) \wedge$$
$$G'(u_3, q_1, q_2, y_3, x_3) \rightarrow \exists N\, G'(u_3, q_1, q_2, N, x_1)$$

This tgd implements the sirup rule. There is also an egd in Σ_t that detects whether δ has been produced:

$$O'(u_1, u_2) \wedge O'(u_2, u_3) \wedge G'(u_3, q_1, q_2, y_1, x) \wedge$$
$$W'(s_1, s_2, x) \wedge W'(s_2, s_3, w) \rightarrow x = w$$

Recall that in general, P is a rule of the form $A(x_1, ..., x_k) \leftarrow \mathbf{Q}_1(\mathbf{u_1}), \ldots, \mathbf{Q}_n(\mathbf{u_n})$, where: x_i, $1 \leq i \leq k$, are variables that occur among the $\mathbf{u_i}$s, and $\mathbf{u_i}$; $0 \leq i \leq n$, are vectors of variables; A occurs among $\mathbf{Q}_1, ..., \mathbf{Q}_n$; and the predicate symbols represented by $\mathbf{Q}_1, ..., \mathbf{Q}_n$ are not necessarily all distinct. Our first tgd in Σ_t is a tgd that implements the sirup rule and is generally of the form:

$$O'(u_1, u_2) \wedge ... \wedge O'(u_{m-1}, u_m) \wedge O'(u_m, x) \wedge$$
$$G'(\mathbf{q_1}, z_1^1, z_2^1, y_1, x_1^1) \wedge \cdots \wedge G'(\mathbf{q_1}, z_{k_1}^1, z_{k_1+1}^1, y_1, x_{k_1}^1) \wedge$$
$$G'(\mathbf{q_2}, z_1^2, z_2^2, y_2, x_1^2) \wedge \cdots \wedge G'(\mathbf{q_2}, z_{k_2}^1, z_{k_2+1}^2, y_2, x_{k_2}^2) \wedge$$
$$\ldots$$
$$G'(\mathbf{q_n}, z_1^n, z_2^n, y_n, x_1^n) \wedge \cdots \wedge G'(\mathbf{q_n}, z_{k_n}^n, z_{k_n+1}^n, y_n, x_{k_n}^n) \wedge$$
$$\rightarrow$$
$$\exists N\, G'(x, z_1^j, z_2^j, N, w_1) \wedge \cdots \wedge G'(x, z_k^j, z_{k+1}^j, N, w_k)$$

The first line above instantiates the variables $u_1, ..., u_m$ to the ordering of our m distinct relations, and binds x to "A". Each subsequent line on the left-hand side of the tgd simulates the atoms $\mathbf{Q}_1(\mathbf{u_1}), \ldots, \mathbf{Q}_n(\mathbf{u_n})$ in the body of the sirup rule. In the tgd, k_i denotes the arity of $\mathbf{Q}_i$, $1 \leq i \leq n$. For example, the second line above is about the atom $\mathbf{Q}_1(\mathbf{u_1})$ of the sirup rule. The vector of variables $(x_1^1, ..., x_{k_1}^1)$ denotes the vector of variables $\mathbf{u_1}$, and the symbol $\mathbf{q_1}$ represents the variable u_5 if $\mathbf{Q}_1$ represents the symbol R_5. The variables $y_1, z_1^1, z_2^1, ..., z_{k_1}^1$ are used to ensure that the variables $x_1^1, ..., x_{k_1}^1$ bind to values of the same tuple and in this order.

The right-hand side of the tgd creates a new tuple $(w_1, ..., w_k)$, where w_is are variables among x_i^js, and k is the arity of A. An existentially quantified variable N is used to generate a new tuple identifier for this tuple. Assuming that A occurs as the jth atom in the body of the sirup rule, the variables $z_1^j, ..., z_{k+1}^j$ are from the $j+1$th line in the above tgd. It is easy to verify that this tgd is weakly acyclic because every special edge leads to a node in the dependency graph without any outgoing edges. Hence, there cannot be a cycle that contains a special edge.

The egd constructed in Σ_t has the form:
$O'(u_1, u_2) \wedge \cdots \wedge O'(u_{m-1}, u_m) \wedge O'(u_m, x) \wedge$
$G'(x, z_1, z_2, y_0, x_1) \wedge \cdots \wedge G'(x, z_{k_1}, z_{k_1+1}, y_0, x_k) \wedge$
$W'(s_1, s_2, x_1) \wedge \cdots \wedge W'(s_k, s_{k+1}, x_k) \wedge W'(s_{k+1}, s_{k+2}, v)$
$\rightarrow u_1 = v$

This egd detects whether the tuple $(c_1, ..., c_k)$ is a tuple in A. If the tuple is in A, the egd becomes applicable and it equates the dummy value "d" from the tuple in W' with some other constant and thus causes a failure.

It is easy to see that the reduction can be performed in polynomial time in the size of P, $\mathbf{D}$, δ, and the ground fact. To perform this reduction, we must make a single pass through the input, and record a polynomial amount of information for every tuple in the input. The target dependencies in Σ_t can also be constructed in polynomial time. Next, we show that $P \cup \mathbf{D} \models \delta$ if and only if there is no solution for the instance I under the schema mapping that we have constructed.

Consider the target instance J' that is the result of chasing the source instance I with Σ_{st}. Surely, J' contains $k+1$ tuples in W' that conceptually represent the tuple $(c_1, ..., c_k, d)$ with $k+1$ columns. Next, consider a run of the sirup rule. It is easy to see that whenever the sirup rule is applied, we can make a corresponding chase step with the tgd in Σ_t. Hence, in effect, the tgd simulates the behavior of the sirup rule. If the fact δ is produced by the sirup rule at some stage, then in the relation G', k tuples will be created that conceptually represent the tuple $A(c_1, ..., c_k)$. Hence, the egd in Σ_t will become applicable and the chase will fail; therefore, there is no solution.

For the converse, since there is no solution, there must be a failing chase sequence. Suppose again that J' is the result of chasing the source instance I with Σ_{st}. As before, J' contains $k+1$ tuples in W' that conceptually represent the tuple $(c_1, ..., c_k, d)$ with $k+1$ columns. Since Σ_{st} consists of full tgds that copy the source relations, we have, by construction, that J' is conceptually a copy of the instance $\mathbf{D}$ and δ. It is easy to see that if the tgd in Σ_t can

be applied, then the sirup rule can be applied as well, and both will produce the same tuple. In the former case, however, the tuple is encoded as k separate tuples in the relation G'. Since there is no solution, the chase must fail due to the egd in Σ_t. This can only happen if there are k tuples in G' that represents the tuple $A(c_1, ..., c_k)$. In other words, we have that $P \cup \mathbf{D} \models \delta$. $\square$

4.3.2 coNP-Completeness

Our next result shows that the existence-of-solutions problem may be coNP-hard even when the schemas are held fixed and each consists of a single relation, and Σ_{st} is fixed and consists of only a single full s-t tgd. Furthermore, our reduction constructs only a single egd in Σ_t to arrive at this lower bound. Thus, a slight extension from a fixed schema mapping immediately gives us the coNP-hard lower bound.

Theorem 4.3.7 *There exist a fixed source schema $\mathbf{S}^*$, a fixed target schema $\mathbf{T}^*$, and a fixed set Σ^*_{st} with the following properties:*

(1) *Each schema consists of a single relational symbol.*

(2) *Σ^*_{st} consists of a single full s-t tgd.*

(3) *The existence-of-solutions problem for $\mathcal{C}^*$ is* coNP*-hard, where $\mathcal{C}^*$ is the class of all schema mappings $(\mathbf{S}^*, \mathbf{T}^*, \Sigma^*_{st}, \Sigma_t)$ such that Σ_t consists of a single egd.*

Proof: We give a reduction from 3SAT. The source schema $\mathbf{S}$ and target schema $\mathbf{T}$ each contain a single relation of arity 5, known as D and D', respectively. There is only a single full s-t tgd $D(u, v, x, y, z) \rightarrow D'(u, v, x, y, z)$ in Σ_{st} that copies D to D'. Given a 3SAT formula φ with n clauses, we let $C_1, C_2, \ldots, C_n$ denote the n clauses. We create the instance $I(\varphi)$ as follows. For each clause C_i, $1 \leq i \leq n$, we create seven tuples in D. For example, if C_i is the clause $(x \vee y \vee \neg z)$, then the following seven tuples are created in D: (C_i,C_{i+1},1,1,1), (C_i,C_{i+1},1,1,0), (C_i,C_{i+1},1,0,1), (C_i,C_{i+1},0,1,1), (C_i,C_{i+1},1,0,0), (C_i,C_{i+1},0,1,0), (C_i,C_{i+1},0,0,0). There is one tuple for every satisfying assignment for C_i. The last three columns contain the satisfying assignment and the first two columns of every tuple contain the values C_i and C_{i+1}, to denote that this satisfying assignment belongs to the clause C_i. Hence, the tuple $(C_i, C_{i+1}, 0, 0, 1)$ does not occur in D since $(0, 0, 1)$ is not a satisfying assignment for C_i. After generating seven tuples for each clause, we add a "dummy" tuple to D: $(C_{n+1}, C_{n+1}, d, d, d)$. The value "$d$" is different from any other value in the relation D.

Finally, we construct an egd $\phi(\mathbf{x}) \rightarrow x_1 = x_2$ for Σ_t based on φ. The left-hand side $\phi(\mathbf{x})$ is constructed as follows: There is a relational atom $D'(u_{n+1}, u_{n+1}, w, w, w)$ in $\phi(\mathbf{x})$, where w is a fresh variable not used anywhere else in $\phi(\mathbf{x})$. Then, for each clause C_i there is a relational atom $D'(u_i, u_{i+1}, x_i, y_i, z_i)$, where x_i, y_i and z_i are variables that represent the literals in C_i. The right-hand side of the egd is $x = w$, where x is one of the variables representing a literal that occur in $\phi(\mathbf{x})$. As an example, suppose φ is the 3SAT formula $(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$. The following egd $D'(u_1, u_2, x_1, x_2, x_3) \wedge D'(u_2, u_3, x_2, x_3, x_4) \wedge D'(u_3, u_3, w, w, w) \rightarrow x_1 = w$ will be constructed.

Clearly, the schema mapping $\mathcal{D}(\varphi)$ and the source instance $I(\varphi)$ can be constructed in polynomial time in the size of φ. It is also easy to see that there is no solution for $I(\varphi)$ under $\mathcal{D}(\varphi)$ if and only if φ is satisfiable. □

Theorem 4.2.3 and Theorem 4.3.7 yield the following corollary.

Corollary 4.3.8 *Let $\mathbf{S}^*$ be a fixed source schema and $\mathbf{T}^*$ a fixed target schema. Let $\mathcal{C}(\mathbf{S}^*, \mathbf{T}^*)$ be the class of all schema mappings $(\mathbf{S}^*, \mathbf{T}^*, \Sigma_{st}, \Sigma_t)$, where Σ_{st} is a set of full s-t tgds, and Σ_t is the union of a set of egds with a set of full tgds. The existence-of-solutions problem for $\mathcal{C}(\mathbf{S}^*, \mathbf{T}^*)$ is* coNP*-complete.*

A slight variation of the proof of Theorem 4.3.7 would yield the same lower bound, where Σ_t is now kept fixed, while Σ_{st} is allowed to vary but consists of only full s-t tgds.

Theorem 4.3.9 *There exist a fixed source schema $\mathbf{S}^*$, a fixed target schema $\mathbf{T}^*$, and a fixed set Σ_t^* with the following properties:*

(1) *Each schema consists of a single relation symbol.*

(2) *Σ_t^* consists of a single egd.*

(3) *The existence-of-solutions problem for $\mathcal{C}^*$ is* coNP*-hard, where $\mathcal{C}^*$ is the class of all schema mappings $(\mathbf{S}^*, \mathbf{T}^*, \Sigma_{st}, \Sigma_t^*)$ such that Σ_{st} consists of a single full s-t tgd.*

Proof: We use the same source and target relations used in the proof of Theorem 4.3.7. We also construct the same source instance used in that proof. For Σ_{st}, we create a full s-t tgd $\phi(\mathbf{x}) \rightarrow D'(u_{n+1}, u_{n+1}, w, w, w)$, where u_{n+1} and w are variables that occur in $\mathbf{x}$, and $\phi(\mathbf{x})$ is the same as the left-hand side of the egd constructed in the proof of Theorem 4.3.7, except that we use the relation symbol D instead of D'. More precisely, $\phi(\mathbf{x})$ is constructed as follows: There is a

relational atom $D(u_{n+1}, u_{n+1}, w, w, w)$ in $\phi(\mathbf{x})$, where w is a variable not used anywhere else in $\phi(\mathbf{x})$. For each clause C_i, $1 \leq i \leq n$, there is a relational atom $D(u_i, u_{i+1}, x_i, y_i, z_i)$ in $\phi(\mathbf{x})$, where x_i, y_i, and z_i are variables that represent the literals in C_i. As an example, suppose the 3SAT formula $(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$. Then, the following s-t tgd is constructed: $D(u_1, u_2, x_1, x_2, x_3) \wedge D(u_2, u_3, x_2, x_3, x_4) \wedge D(u_3, u_3, w, w, w) \rightarrow D'(u_3, u_3, w, w, w)$. Finally, Σ_t consists of a single fixed egd $D'(u, u, w, w, w) \rightarrow u = w$. It is easy to see that the 3SAT formula is satisfiable if and only if there is no solution for the source instance under the schema mapping. □

4.4 Concluding Remarks

4.4.1 Summary of Results

The results presented here shed light on both the data complexity and the combined complexity of data exchange for schema mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\mathbf{st}}, \Sigma_{\mathbf{t}})$ in which Σ_{st} is a set of s-t tgds and Σ_t is a set of target egds and target tgds. Earlier work on the data complexity of data exchange had shown that the existence-of-solutions problem is polynomial-time solvable, assuming that the target tgds form a weakly acyclic set. Regarding the combined complexity of data exchange, we showed that there is a provable exponential gap between the data complexity and the combined complexity of the existence-of-solutions problem for schema mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\mathbf{st}}, \Sigma_{\mathbf{t}})$ in which Σ_{st} is a set of s-t tgds and Σ_t is the union of a set of target egds with a weakly acyclic set of target tgds.

4.4.2 Future Work

We have shown that in restricted settings, the existence-of-solutions problem can be coNP-complete or EXPTIME-complete. For the general case, however, we do not have a tight lower bound to match the 2EXPTIME upper bound. To make this bound tight, we need either to produce a new EXPTIME algorithm for solving the existence-of-solutions problem in general, or to produce a reduction from a known 2EXPTIME-complete problem.

In general there may be infinitely many solutions to a data exchange instance. Fagin et al. presented, in [9], the concept of a *universal solution*, which is a solution that makes the smallest number of assumptions about the data. Given a schema mapping $\mathcal{M}$ and an in-

stance I, a universal solution J for I is a solution for I that has a homomorphism to every other solution for I. They then proved that a universal solution could be computed in polynomial time.

There may still be infinitely many universal solutions, and so in [10] Fagin et al. discussed using the core of the universal solutions as the desired goal. The core of a relational structure $\mathbf{A}$ is a structure $\mathbf{B}$ such that $\mathbf{B} \subseteq \mathbf{A}$, there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$, and there is no proper substructure $\mathbf{C}$ of $\mathbf{B}$ such that there is a homomorphism from $\mathbf{B}$ to $\mathbf{C}$. This means that the core of $\mathbf{A}$ is the smallest substructure of $\mathbf{A}$ that is homomorphically equivalent to $\mathbf{A}$. It turns out that if you compute the core of any two universal solutions, you will get isomorphic structures. Thus we have a unique case of the universal solutions of an instance of data exchange. There are a number of results about the computation complexity of computing the core, but the general complexity is still unknown.

Bibliography

[1] J. Barwise. On Moschovakis closure ordinals. *Journal of Symbolic Logic*, 42:292–296, 1977.

[2] A. A. Bulatov. A dichotomy theorem for constraints on a three-element set. In *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 649–658, Washington, DC, USA, 2002. IEEE Computer Society.

[3] S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, NY, USA, 1971. ACM Press.

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

[5] R. Dechter. From local to global consistency. *Artificial Intelligence*, 55:87–107, 1992.

[6] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[7] A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints. In *International Conference on Database Theory (ICDT)*, pages 225–241, 2003.

[8] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer - Verlag, 1998.

[9] R. Fagin, Ph. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science (TCS)*, 336(1):89–124, 2005.

[10] R. Fagin, Ph. G. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. *ACM Transactions on Database Systems (TODS)*, 30(1):174–210, 2005.

[11] T. Feder and M. Y. Vardi. Monotone monadic SNP and constraint satisfaction. In *STOC '93: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 612–622, New York, NY, USA, 1993. ACM Press.

[12] 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, 1998. Preliminary version in *Proc. 25th ACM Symp. on Theory of Computing*, May 1993, pp. 612–622.

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

[14] G. Gottlob and C. Papadimitriou. On the complexity of single-rule datalog queries. *Information and Computation*, 183(1):104–122, 2003.

[15] M. Grohe. Equivalence in finite-variable logics is complete for polynomial time. *Combinatorica*, 19(4):507–523, 1999.

[16] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM (JACM)*, 44(4):527–548, 1997.

[17] T. Kasai, A. Adachi, and S. Iwata. Classes of pebble games and complete problems. *SIAM Journal of Computing*, 8(4):574–586, 1979.

[18] S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45(3):275–286, 1990.

[19] Ph. G. Kolaitis and J. Panttaja. On the complexity of existential pebble games. In *Proceedings of Computer Science Logic CSL '03, Lecture Notes in Computer Science 2803*, pages 314–329. Springer, 2003.

[20] Ph. G. Kolaitis, J. Panttaja, and W. Tan. The complexity of data exchange. In *Proceedings of the 25th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'06)*, pages 30–39, 2006.

[21] Ph. G. Kolaitis and M. Y. Vardi. On the expressive power of Datalog: Tools and a case study. *Journal of Computer and System Sciences*, 51:110–134, 1995.

[22] Ph. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61:302–332, 2000.

[23] Ph. G. Kolaitis and M. Y. Vardi. A game-theoretic approach to constraint satisfaction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 175–181, 2000.

[24] Ph. G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proceedings of the 17th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)*, pages 205–213, 1998.

[25] E. Pezzoli. Computational complexity of Ehrenfeucht-Fraïssé games on finite structures. In *Computer Science Logic. 12th International Workshop, CSL'98.*, pages 159–170, 1999.

[26] T.J. Schaefer. The Complexity of Satisfiability Problems. In *ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978.