Data Exchange: Getting to the Core

Ronald Fagin
IBM Almaden Research Center
fagin@almaden.ibm.com

Phokion G. Kolaitis*
UC Santa Cruz
kolaitis@cse.ucsc.edu

Lucian Popa
IBM Almaden Research Center
lucian@almaden.ibm.com

ABSTRACT

Data exchange is the problem of taking data structured under a source schema and creating an instance of a target schema that reflects the source data as accurately as possible. Given a source instance, there may be many solutions to the data exchange problem, that is, many target instances that satisfy the constraints of the data exchange problem. In an earlier paper, we identified a special class of solutions that we call *universal*. A universal solution has homomorphisms into every possible solution, and hence is a "most general possible" solution. Nonetheless, given a source instance, there may be many universal solutions. This naturally raises the question of whether there is a "best" universal solution, and hence a best solution for data exchange. We answer this question by considering the well-known notion of the *core* of a structure, a notion that was first studied in graph theory, but has also played a role in conjunctive-query processing. The core of a structure is the smallest substructure that is also a homomorphic image of the structure. All universal solutions have the same core (up to isomorphism); we show that this core is also a universal solution, and hence the smallest universal solution. The uniqueness of the core of a universal solution together with its minimality make the core an ideal solution for data exchange. Furthermore, we show that the core is the best among all universal solutions for answering unions of conjunctive queries with inequalities. After this, we investigate the computational complexity of producing the core. Well-known results by Chandra and Merlin imply that, unless $\mathrm{P} = \mathrm{NP}$, there is no polynomial-time algorithm that, given a structure as input, returns the core of that structure as output. In contrast, in the context of data exchange, we identify natural and fairly broad conditions under which there are polynomial-time algorithms for computing the core of a universal solution. Finally, we analyze the computational complexity of the following decision problem that underlies the computation of cores: given two graphs G and H, is H the core of G? Earlier results imply that this problem is both NP-hard and coNP-hard. Here, we pinpoint its exact complexity by establishing that it is a DP-complete problem.

* Partially supported by NSF Grant IIS-9907419.

PODS 2003, June 9-12, 2003, San Diego, CA.

1. Introduction and Summary of Results

The data exchange problem Data exchange is the problem of materializing an instance that adheres to a target schema, given an instance of a source schema and a specification of the relationship between the source schema and the target schema. This problem arises in many tasks requiring data to be transferred between independent applications that do not necessarily adhere to the same data format (or schema). The importance of data exchange was recognized a long time ago; in fact, an early data exchange system was EXPRESS [20] from the 1970's, whose main functionality was to convert data between hierarchical schemas. The need for data exchange has steadily increased over the years and, actually, has become more pronounced in recent years, with the proliferation of web data in various formats and with the emergence of e-business applications that need to communicate data yet remain autonomous. The data exchange problem is related to the data integration problem in the sense that both problems are concerned with management of data stored in heterogeneous formats. The two problems, however, are different for the following reasons. In data exchange, the main focus is on actually *materializing* a target instance that reflects the source data as accurately as possible; this can be a serious challenge, due to the inherent underspecification of the relationship between the source and the target. In contrast, a target instance need not be materialized in data integration; the main focus there is on answering queries posed over the target schema using views that express the relationship between the target and source schemas.

In a previous paper [9], we formalized the data exchange problem and embarked on an in-depth investigation of the foundational and algorithmic issues that surround it. Our work has been motivated by practical considerations arising in the ongoing development of Clio [16, 19], a prototype system for schema mapping and data exchange between autonomous applications. A data exchange setting is a quadruple $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S}$ is the source schema, $\mathbf{T}$ is the target schema, Σ_{st} is a set of source-to-target dependencies that express the relationship between $\mathbf{S}$ and $\mathbf{T}$, and Σ_t is a set of dependencies that express constraints on $\mathbf{T}$. Such a setting gives rise to the following *data exchange problem*: given an instance I over the source schema $\mathbf{S}$, find an instance J over the target schema $\mathbf{T}$ such that I together with J satisfy the source-to-target dependencies Σ_{st}, and J satisfies the target dependencies Σ_t. Such an instance J is called a *solution* for I in the data exchange setting. In general, many different solutions for an instance I may exist. Thus, the question is: which solution should one choose to materialize, so that it reflects the source data as accurately as possible? Moreover, can such a solution be efficiently computed?

In [9], we investigated these issues for data exchange settings in which $\mathbf{S}$ and $\mathbf{T}$ are relational schemas, Σ_{st} is a set of tuple-generating dependencies (tgds) between $\mathbf{S}$ and $\mathbf{T}$, and Σ_t is a set of tgds and equality-generating dependencies (egds) on $\mathbf{T}$. We isolated a class of solutions, called *universal solutions*, possessing good properties that justify selecting them as the semantics of the data exchange problem. Specifically, universal solutions have homomorphisms into every possible solution; in particular, they have homomorphisms into each other, and thus are homomorphically equivalent. Universal solutions are the most general among all solutions and, in a precise sense, they represent the entire space of solutions. Moreover, as we shall explain shortly, universal solutions can be used to compute the "certain answers" of queries q that are unions of conjunctive queries over the target schema. The set *certain*(q, I) of *certain answers* of a query q over the target schema, with respect to a source instance I, consists of all tuples that are in the intersection of all $q(J)$'s, as J varies over all solutions for I (here, $q(J)$ denotes the result of evaluating q on J). The notion of the certain answers originated in the context of incomplete databases (see [21] for a survey). Moreover, the certain answers have been used for query answering in data integration [14]. In the same data integration context, Abiteboul and Duschka [1] studied the complexity of computing the certain answers.

We showed [9] that the certain answers of unions of conjunctive queries can be obtained by simply evaluating these queries on some arbitrarily chosen universal solution. We also showed that, under fairly general, yet practical, conditions, a universal solution exists whenever a solution exists. Furthermore, we showed that when these conditions are satisfied, there is a polynomial-time algorithm for computing a *canonical* universal solution; this algorithm is based on the classical chase procedure [3, 15].

Data exchange with cores Even though they are homomorphically equivalent to each other, universal solutions need not be unique. In other words, in a data exchange setting, there may be many universal solutions for a given source instance I. Thus, it is natural to ask: what makes a universal solution "better" than another universal solution? Is there a "best" universal solution and, of course, what does "best" really mean? If there is a "best" universal solution, can it be efficiently computed?

The present paper addresses these questions and offers answers that are based on using *minimality* as a key criterion for what constitutes the "best" universal solution. Although universal solutions come in different sizes, they all share a unique (up to isomorphism) common "part", which is nothing else but the *core* of each of them, when they are viewed as relational structures. By definition, the core of a structure is the smallest substructure that is also a homomorphic image of the structure. The concept of the core originated in graph theory, where a number of results about its properties have been established (see, for instance, [12]). Moreover, in the early days of database theory, Chandra and Merlin [4] realized that the core of a structure is useful in conjunctive-query processing. Indeed, since evaluating joins is the most expensive among the basic relational algebra operations, one of the most fundamental problems in query processing is the join-minimization problem: given a conjunctive query q, find an equivalent conjunctive query involving the smallest possible number of joins. In turn, this problem amounts to computing the core of the relational instance $\mathbf{D}_q$ that is obtained from q by putting a fact into $\mathbf{D}_q$ for each conjunct of q (see [2, 4, 13]).

Consider a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ in which Σ_{st} is a set of tgds and Σ_t is a set of tgds and egds. Since all universal solutions for a source instance I are homomorphically equivalent, it is easy to see that their cores are isomorphic. Moreover, we show in this paper that the core of a universal solution for I is itself a solution for I. It follows that the core of the universal solutions for I is the *smallest* universal solution for I, and thus an ideal candidate for the "best" universal solution, at least in terms of the space required to materialize it.

We further justify the selection of the core as the "best" universal solution by establishing its usefulness in answering queries over the target schema $\mathbf{T}$. Let J_0 be the core of all universal solutions for a source instance I. As discussed earlier, since J_0 is itself a universal solution for I, the certain answers of conjunctive queries over $\mathbf{T}$ can be obtained by simply evaluating them on J_0. In [9], however, it was shown that there are simple conjunctive queries with inequalities $\neq$ such that evaluating them on a universal solution produces a *proper superset* of the set of certain answers for I. Nonetheless, here we show that evaluating conjunctive queries with inequalities on the core J_0 of the universal solutions yields the *best approximation* (i.e., smallest superset) of the set of the certain answers, among all universal solutions. Indeed, we show that if q is a union of conjunctive queries with inequalities, then the set of those tuples in $q(J_0)$ whose entries are elements from the source instance I is equal to the intersection of all $q(J)$'s, where J varies over all universal solutions for I.

Having established the preceding good properties of the core in data exchange, we then address the issue of how hard it is to compute the core of a universal solution. Chandra and Merlin [4] showed that join minimization is an NP-hard problem by pointing out that a graph $\mathbf{G}$ is 3-colorable if and only if the 3-element clique $\mathbf{K}_3$ is the core of the disjoint sum $\mathbf{G} \oplus \mathbf{K}_3$ of $\mathbf{G}$ with $\mathbf{K}_3$. From this, it follows that, unless $\mathrm{P} = \mathrm{NP}$, there is no polynomial-time algorithm that, given a structure as input, outputs its core. At first sight, this result casts doubts on the tractability of computing the core of a universal solution. For data exchange, however, we give natural and fairly broad conditions under which there are polynomial-time algorithms for computing the cores of universal solutions. Specifically, we show that there is a polynomial-time algorithm for computing the core in a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ in which Σ_{st} is an arbitrary set of tgds, but Σ_t is empty, that is, there are no target constraints. We then extend this result by showing that it holds even when Σ_t is an arbitrary set of egds. We conjecture that there is a polynomial-time algorithm for computing the core in even broader data exchange settings in which Σ_t may also contain tgds, and we leave this as an open problem. We also analyze the computational complexity of the following decision problem, called CORE IDENTIFICATION, which underlies the computation of cores: given two graphs $\mathbf{G}$ and $\mathbf{H}$, is $\mathbf{H}$ the core of $\mathbf{G}$? As seen above, the results by Chandra and Merlin [4] imply that this problem is NP-hard. Later on, Hell and Nešetřil [12] showed that deciding whether a graph $\mathbf{G}$ is its own core is a coNP-complete problem; in turn, this implies that CORE IDENTIFICATION is a coNP-hard problem. Here, we pinpoint the exact computational complexity of CORE IDENTIFICATION by showing that it is a DP-complete problem, where DP is the class of decision problems that can be written as the intersection of an NP-problem and a coNP-problem.

2. Preliminaries

This section reviews the main definitions related to data exchange that are needed for the results of this paper. In presenting the necessary definitions, we follow closely our earlier paper [9].

2.1 The Data Exchange Problem

A *schema* is a finite sequence $\mathbf{R} = \langle R_1, \ldots, R_k \rangle$ of relation symbols, each of a fixed arity. An *instance I (over the schema $\mathbf{R}$)* is a sequence $\langle R_1^I, \ldots, R_k^I \rangle$ that associates each relation symbol R_i with a relation R_i^I of the same arity as R_i. We shall often abuse the notation and use R_i to denote both the relation symbol and the relation R_i^I that interprets it. We may refer to R_i^I as the R_i *relation of* I. Given a tuple t occurring in a relation R, we denote by $R(t)$ the association between t and R, and call it a *fact*. If $\mathbf{R}$ is a schema, then a *dependency over* $\mathbf{R}$ is a sentence in some logical formalism over $\mathbf{R}$.

Let $\mathbf{S} = \langle S_1, \ldots, S_n \rangle$ and $\mathbf{T} = \langle T_1, \ldots, T_m \rangle$ be two schemas with no relation symbols in common. We refer to $\mathbf{S}$ as the *source* schema and to the S_i's as the *source* relation symbols. We refer to $\mathbf{T}$ as the *target* schema and to the T_j's as the *target* relation symbols. We denote by $\langle \mathbf{S}, \mathbf{T} \rangle$ the schema $\langle S_1, \ldots, S_n, T_1, \ldots, T_m \rangle$. Instances over $\mathbf{S}$ will be called *source* instances, while instances over $\mathbf{T}$ will be called *target* instances. If I is a source instance and J is a target instance, then we write $\langle I, J \rangle$ for the instance K over the schema $\langle \mathbf{S}, \mathbf{T} \rangle$ such that $S_i^K = S_i^I$ and $T_j^K = T_j^J$, when $1 \leq i \leq n$ and $1 \leq j \leq m$.

A *source-to-target dependency* is, in general, a dependency over $\langle \mathbf{S}, \mathbf{T} \rangle$ of the form $\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \chi_{\mathbf{T}}(\mathbf{x}))$, where $\phi_{\mathbf{S}}(\mathbf{x})$ is a formula, with free variables $\mathbf{x}$, of some logical formalism over $\mathbf{S}$, and $\chi_{\mathbf{T}}(\mathbf{x})$ is a formula, with free variables $\mathbf{x}$, of some logical formalism over $\mathbf{T}$ (these two logical formalisms may be different). We use the notation $\mathbf{x}$ for a vector of variables $x_1, \ldots, x_k$. A *target* dependency is, in general, a dependency over the target schema $\mathbf{T}$ (the formalism used to express a target dependency may be different from those used for the source-to-target dependencies). The source schema may also have dependencies that we assume are satisfied by every source instance. While the source dependencies may play an important role in deriving source-to-target dependencies [19], they do not play any direct role in data exchange, because we take the source instance to be *given*.

DEFINITION 2.1. A *data exchange setting* $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ consists of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, a set Σ_{st} of source-to-target dependencies, and a set Σ_t of target dependencies. The *data exchange problem* associated with this setting is the following: given a finite source instance I, find a finite target instance J such that $\langle I, J \rangle$ satisfies Σ_{st} and J satisfies Σ_t. Such a J is called a *solution for* I or, simply, a *solution* if the source instance I is understood from the context. ∎

For most practical purposes, and for most of the results of this paper (except for Proposition 2.7) each source-to-target dependency in Σ_{st} is a *tuple generating dependency (tgd)* [3] of the form

$$\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})),$$

where $\phi_{\mathbf{S}}(\mathbf{x})$ is a conjunction of atomic formulas over $\mathbf{S}$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathbf{T}$. Moreover, each target dependency in Σ_t is either a tgd, of the form

$$\forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})),$$

or an *equality-generating dependency (egd)* [3], of the form

$$\forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_1 = x_2)).$$

In these dependencies, $\phi_{\mathbf{T}}(\mathbf{x})$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over $\mathbf{T}$, and x_1, x_2 are among the variables in $\mathbf{x}$. As in [9], we will drop the universal quantifiers in front of a dependency, and implicitly assume such quantification. However, we will write down all the existential quantifiers.

Source-to-target tgds are a natural and powerful language for expressing the relationship between a source schema and a target schema. Such dependencies are automatically derived and used as representation of a schema mapping in the Clio system [19]. Furthermore, data exchange settings with tgds as source-to-target dependencies include as special cases both LAV and GAV data integration systems in which the views are sound and defined by conjunctive queries (see Lenzerini's tutorial [14] for a detailed discussion of LAV and GAV data integration systems and sound views).

Indeed, a *LAV data integration system with sound views defined by conjunctive queries* is a special case of a data exchange setting ($\mathbf{S}$, $\mathbf{T}$, Σ_{st}, Σ_t), in which $\mathbf{S}$ is the source schema (consisting of the views, in LAV terminology), $\mathbf{T}$ is the target schema (or global schema, in LAV terminology), the set Σ_t of target dependencies is empty, and each source-to-target tgd in Σ_{st} is of the form $S(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$, where S is a single relation symbol of the source schema $\mathbf{S}$ (a view, in LAV terminology) and $\psi_{\mathbf{T}}$ is a conjunction of atomic formulas over the target schema $\mathbf{T}$. A GAV setting is similar, but the tgds in Σ_{st} are of the form $\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow T(\mathbf{x})$, where T is a single relation symbol over the target schema $\mathbf{T}$ (a view, in GAV terminology), and $\phi_{\mathbf{S}}$ is a conjunction of atomic formulas over the source schema $\mathbf{S}$. Since, in general, a source-to-target tgd relates a conjunctive query over the source schema to a conjunctive query over the target schema, a data exchange setting is strictly more expressive than LAV or GAV, and in fact it can be thought of as a GLAV (*global-and-local-as-view*) system [10, 14]. These similarities between data integration and data exchange notwithstanding, the main difference between the two is that in data exchange we have to actually materialize a finite target instance that best reflects the given source instance. In data integration no such exchange of data is required; the target can remain virtual.

In general there may be multiple solutions for a given data exchange problem. The following example illustrates this issue and raises the question of which solution to choose to materialize.

EXAMPLE 2.2. *[9]* Consider a data exchange problem in which the source schema has three relation symbols P, Q, R, each of them with attributes A, B, C, while the target schema has one relation symbol T also with attributes A, B, C. We assume that $\Sigma_t = \emptyset$. The source-to-target tgds and the source instance are:

$$\begin{aligned} \Sigma_{st}: \quad & P(a,b,c) \rightarrow \exists Y \exists Z\ T(a,Y,Z) \\ & Q(a,b,c) \rightarrow \exists X \exists U\ T(X,b,U) \\ & R(a,b,c) \rightarrow \exists V \exists W\ T(V,W,c) \\ I = \quad & \{P(a_0,b_0',c_0'), Q(a_0'',b_0,c_0''),\ R(a_0''',b_0''',c_0)\} \end{aligned}$$

Since the tgds in Σ_{st} do not completely specify the target instance, there are multiple solutions that are consistent with the specification. One solution is:

$$J = \{T(a_0, Y_0, Z_0), T(X_0, b_0, U_0), T(V_0, W_0, c_0)\},$$

where $X_0, Y_0, \ldots$ represent "unknown" values, that is, values that

do not occur in the source instance. Such values are called *labeled nulls* and are to be distinguished from the values occuring in the source instance, which are called *constants*. Instances with constants and labeled nulls are not specific to data exchange. They have long been considered, in various forms, in the context of incomplete or indefinite databases (see [21]) as well as in the context of data integration (see [11, 14]). For the current example, the following instances are solutions as well:

$$J_1 = \{T(a_0, b_0, c_0)\} \qquad J_2 = \{T(a_0, b_0, Z_1), T(V_1, W_1, c_0)\}$$

In the above, Z_1, V_1 and W_1 are labeled nulls. Note that J_1 does not use labeled nulls, but uses constants to witness the existentially quantified variables of the tgds. ∎

Next, we review the notion of universal solutions, proposed in [9] as the most general solutions.

2.2 Universal Solutions

We denote by $\underline{\mathsf{Const}}$ the set (possibly infinite) of all values that occur in source instances, and as before we call them *constants*. We also assume an infinite set $\underline{\mathsf{Var}}$ of values, called *labeled nulls*, such that $\underline{\mathsf{Var}} \cap \underline{\mathsf{Const}} = \emptyset$. We reserve the symbols $I, I', I_1, I_2, \ldots$ for instances over the source schema $\mathbf{S}$ and with values in $\underline{\mathsf{Const}}$. We reserve the symbols $J, J', J_1, J_2, \ldots$ for instances over the target schema $\mathbf{T}$ and with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$. Moreover, we require that solutions of a data exchange problem have their values drawn from $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$. If $\mathbf{R} = \langle R_1, \ldots, R_k \rangle$ is a schema and K is an instance over $\mathbf{R}$ with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$, then $\underline{\mathsf{Const}}(K)$ denotes the set of all constants occurring in relations in K, and $\underline{\mathsf{Var}}(K)$ denotes the set of labeled nulls occurring in relations in K.

DEFINITION 2.3. Let K_1 and K_2 be two instances over $\mathbf{R}$ with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$.

1. A *homomorphism* $h : K_1 \to K_2$ is a mapping from $\underline{\mathsf{Const}}(K_1) \cup \underline{\mathsf{Var}}(K_1)$ to $\underline{\mathsf{Const}}(K_2) \cup \underline{\mathsf{Var}}(K_2)$ such that: (1) $h(c) = c$, for every $c \in \underline{\mathsf{Const}}(K_1)$; (2) for every fact $R_i(t)$ of K_1, we have that $R_i(h(t))$ is a fact of K_2 (where, if $t = (a_1, \ldots, a_s)$, then $h(t) = (h(a_1), \ldots, h(a_s))$).
2. K_1 is *homomorphically equivalent* to K_2 if there are homomorphisms $h : K_1 \to K_2$ and $h' : K_2 \to K_1$. ∎

DEFINITION 2.4 (UNIVERSAL SOLUTION). Consider a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$. If I is a source instance, then a *universal solution for* I is a solution J for I such that for every solution J' for I, there exists a homomorphism $h : J \to J'$. ∎

EXAMPLE 2.5. The solutions J_1 and J_2 in Example 2.2 are not universal. In particular, there is no homomorphism from J_1 to J and there is no homomorphism from J_2 to J. Hence, the solutions J_1 and J_2 contain "extra" information that was not required by the specification. In contrast, it can easily be shown that J has homomorphisms to every solution. Thus, J is universal. ∎

Universal solutions possess good properties that justify selecting them (as opposed to arbitrary solutions) for the semantics of the data exchange problem. A universal solution is more general than an arbitrary solution because, by definition, it can be homomorphically mapped into that solution. Universal solutions have, also by their definition, homomorphisms to each other and, thus, are homomorphically equivalent.

Computing universal solutions In [9], we addressed the question of how to check the existence of a universal solution and how to compute one, if one exists. In particular, we identified fairly general, yet practical, conditions that guarantee that universal solutions exist whenever solutions exist. Moreover, we showed that there is a polynomial-time algorithm for computing a *canonical* universal solution, if a solution exists; this algorithm is based on the classical chase procedure. The following result summarizes these findings.

THEOREM 2.6. *[9] Assume a data exchange setting where Σ_{st} is a set of tgds, and Σ_t is the union of a weakly acyclic set of tgds with a set of egds.*

1. *The existence of a solution can be checked in polynomial time.*
2. *A universal solution exists if and only if a solution exists.*
3. *If a solution exists, then a universal solution can be produced in polynomial time using the chase.*

The notion of a *weakly acyclic set of tgds* first arose in a conversation between the third author and A. Deutsch in 2001. It was then independently used in [8] and in [9] (in the former paper, under the term *constraints with stratified-witness*). This class guarantees the termination of the chase and is quite broad, as it includes both sets of full tgds [3] and sets of acyclic inclusion dependencies [6]. We note that, when the set Σ_t of target constraints is empty, a universal solution always exists and a canonical one is constructible in polynomial time by chasing $\langle I, \emptyset \rangle$ with Σ_{st}. In the Example 2.2, the instance J is such a canonical universal solution. If the set Σ_t of target constraints contains egds, then it is possible that no universal solution exists (and hence no solution exists, either, by the above theorem). This occurs (see [9]) when the chase fails by attempting to identify two constants while trying to apply some egd of Σ_t. If the chase does not fail, then the result of chasing $\langle I, \emptyset \rangle$ with $\Sigma_{st} \cup \Sigma_t$ is a canonical universal solution.

Certain answers on universal solutions In a data exchange setting, there may be many possible solutions for the target instance. Hence, there is a question of what is the result of answering queries over the target schema. In [9], and following work on data integration, we adopted the notion of the certain answers as the semantics of query answering. Recall that the set *certain*(q, I) of the certain answers of q with respect to a source instance I is the set of tuples that appear in $q(J)$ for *every* solution J. Moreover, if J is an arbitrary solution, let us denote by $q(J)_\downarrow$ the set of all "null-free" tuples in $q(J)$, that is the set of all tuples in $q(J)$ that are formed entirely of constants. The next proposition from [9] shows that null-free evaluation of conjunctive queries on an arbitrarily chosen universal solution gives precisely the set of certain answers. Moreover, universal solutions are the only solutions that have this property.

PROPOSITION 2.7. *[9] Consider a data exchange setting with $\mathbf{S}$ as the source schema, $\mathbf{T}$ as the target schema, and such that the dependencies in the sets Σ_{st} and Σ_t are arbitrary.*

1. *Let q be a union of conjunctive queries over the target schema $\mathbf{T}$. If I is a source instance and J is a universal solution, then certain$(q, I) = q(J)_\downarrow$.*
2. *Let I be a source instance and J be a solution such that for every conjunctive query q over $\mathbf{T}$, we have that certain$(q, I) =$*

$q(J)_{\downarrow}$. *Then J is a universal solution.*

3. Multiple Universal Solutions

We give next a very simple example showing that, even if we restrict our attention to universal solutions instead of arbitrary solutions, there may still be multiple, non-isomorphic, universal solutions for a given data exchange problem. Although these universal solutions are homomorphically equivalent, they have different sizes. Moreover, the example shows that evaluating conjunctive queries with inequalities $\neq$ on universal solutions may not always produce the certain answers. (Contrast this with Proposition 2.7, which deals with the case of unions of conjunctive queries.)

EXAMPLE 3.1. Consider a data exchange problem in which the source schema has one relation symbol S with attributes A, B, while the target schema has one relation symbol T with attributes A, C. We assume that $\Sigma_t = \emptyset$. The source-to-target tgds and the source instance are:

$$\begin{aligned} \Sigma_{st}: \quad & S(x,y) \rightarrow \exists Z\, T(x,Z) \\ I = \quad & \{S(a,b_1), S(a,b_2)\} \end{aligned}$$

Then the following instances are universal solutions:

$$J_1 = \{T(a,Z_1), T(a,Z_2)\}, \qquad J_2 = \{T(a,Z)\},$$

where Z_1, Z_2, and Z are labeled nulls. It is easy to verify that J_1 and J_2 are universal. From a size perspective, we have that J_2 is smaller than J_1. It can be shown that every universal solution must contain at least one tuple of the form (a, Z') for some null Z'. Hence, J_2 has the smallest size among all universal solutions. We argue that J_2, rather than J_1, should be used for data exchange.

By Proposition 2.7, queries that are unions of conjunctive queries give the same answers (the certain answers) when applied to either J_1 or J_2, since both solutions are universal. However, consider the conjunctive query with one inequality:

$$q(x,y) = \exists Z_1 \exists Z_2\, (T(x,Z_1) \wedge T(y,Z_2) \wedge (Z_1 \neq Z_2)).$$

Then q returns $\{(a,a)\}$ when evaluated on J_1, and returns the empty set when evaluated on J_2. In particular, this also shows that (a,a) is not a certain answer; hence, query evaluation on a universal solution (J_1 in this case) may be a strict superset of the set of certain answers (which is the empty set for this example). On the other hand, $q(J_2)$ coincides with the set of certain answers. In general, the exact equality with the set of certain answers may not always be possible, even if we use a smallest universal solution such as J_2; however, we always obtain a *best* approximation, as we shall prove in the next section. This is further justification for selecting J_2 rather than J_1. ∎

Following the above intuition, we define and study, in the next section, *cores* of universal solutions. We prove that, in general, such cores of universal solutions enjoy two good properties. Specifically, they are the smallest universal solutions and they provide the best approximation of the set of certain answers (among all universal solutions). This, combined with the fact that there is only one such universal solution (up to isomorphism), as we shall prove, gives a strong justification for using the core of universal solutions for data exchange.

4. Data Exchange with Cores: Semantics and Query Answering

4.1 Cores and Universal Solutions

We find it convenient to define not only an *instance* (which we defined earlier), but also the closely related notion of a *structure*. The difference is that a structure is defined with a *universe*, whereas the universe of an instance is implicitly taken to be the "active domain", that is, the set of elements that appear in tuples of the instance. Furthermore, unlike target instances in data exchange settings, structures do not necessarily have distinguished elements ("constants") that have to be mapped onto themselves by homomorphisms.

More formally, a *structure* $\mathbf{A}$ (over the schema $\mathbf{R} = \langle R_1, \ldots, R_k \rangle$) is a sequence $\langle A, R_1^{\mathbf{A}}, \ldots, R_k^{\mathbf{A}} \rangle$, where A is a non-empty set, called the *universe*, and that associates each relation symbol R_i with a relation $R_i^{\mathbf{A}}$ of the same arity as R_i. As with instances, we shall often abuse the notation and use R_i to denote both the relation symbol and the relation $R_i^{\mathbf{A}}$ that interprets it. We may refer to $R_i^{\mathbf{A}}$ as the R_i *relation of* $\mathbf{A}$. If A is finite, then we say that the structure is finite. A structure $\mathbf{B} = (B, R_1^{\mathbf{B}}, \ldots, R_k^{\mathbf{B}})$ is a *substructure* of $\mathbf{A}$ if $B \subseteq A$ and $R_i^{\mathbf{B}} \subseteq R_i^{\mathbf{A}}$, for $1 \leq i \leq k$. We say that $\mathbf{B}$ is a *proper substructure* of $\mathbf{A}$ if it is a substructure of $\mathbf{A}$ and at least one of the containments $R_i^{\mathbf{B}} \subseteq R_i^{\mathbf{A}}$, for $1 \leq i \leq k$, is a proper one.

DEFINITION 4.1. A substructure $\mathbf{C}$ of structure $\mathbf{A}$ is called a *core of* A if there is a homomorphism from $\mathbf{A}$ to $\mathbf{C}$, but there is no homomorphism from $\mathbf{A}$ to a proper substructure of $\mathbf{C}$. A structure $\mathbf{C}$ is called a *core* if it is a core of itself, that is, if there is no homomorphism from $\mathbf{C}$ to a proper substructure of $\mathbf{C}$. ∎

Note that $\mathbf{C}$ is a core of $\mathbf{A}$ if and only if $\mathbf{C}$ is a core, $\mathbf{C}$ is a substructure of $\mathbf{A}$, and there is a homomorphism from $\mathbf{A}$ to $\mathbf{C}$. The concept of the core of a graph has been studied extensively in graph theory (see [12]). The next proposition summarizes some basic facts about cores; a proof can be found in [12].

PROPOSITION 4.2. *The following statements hold:*

- *Every finite structure has a core; moreover, all cores of the same finite structure are isomorphic.*
- *Every finite structure is homomorphically equivalent to its core. Consequently, two finite structures are homomorphically equivalent if and only if their cores are isomorphic.*
- *If $\mathbf{C}$ is the core of a finite structure $\mathbf{A}$, then there is a homomorphism $h: \mathbf{A} \rightarrow \mathbf{C}$ such that $h(v) = v$ for every member v of the universe of $\mathbf{C}$.*

In view of Proposition 4.2, if $\mathbf{A}$ is a finite structure, there is a unique (up to isomorphism) core of $\mathbf{A}$, which we denote by $\underline{\mathsf{core}}(\mathbf{A})$.

We can similarly define the notions of a subinstance of an instance and of a core of an instance. We identify the instance with the corresponding structure, where the universe of the structure is taken to be the active domain of the instance, and where we distinguish the constants in the universe. That is, we require that if h is a homomorphism and c is a constant, then $h(c) = c$ (as already defined in Section 2.2). The results about cores of structures will then carry over to cores of instances.

As seen earlier, universal solutions for I are unique up to homomorphic equivalence, but they need not be unique up to isomorphism.

Proposition 4.2, however, implies that their cores are isomorphic; in other words, all universal solutions for I have the same core up to isomorphism. Moreover, if $\underline{\text{core}}(J)$ is a solution for I, then it is also a universal solution for I, since J and $\underline{\text{core}}(J)$ are homomorphically equivalent. In general, if the dependencies Σ_{st} and Σ_t are arbitrary, then the core of a solution to an instance of the data exchange problem need not be a solution. The next result shows, however, that this cannot happen if Σ_{st} is a set of tgds and Σ_t is the union of a set of tgds with a set of egds.

PROPOSITION 4.3. *Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting in which Σ_{st} is a set of tgds and Σ_t is the union of a set of tgds with a set of egds. If I is a source instance and J is a solution for I, then $\underline{\text{core}}(J)$ is a solution for I. Consequently, if J is a universal solution for I, then also $\underline{\text{core}}(J)$ is a universal solution for I.*

Proof: Let $\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ be a tgd in Σ_{st} and $\mathbf{a} = (a_1, \ldots, a_n)$ a tuple of constants such that $I \models \phi_{\mathbf{S}}(\mathbf{a})$. Since J is a solution for I, there is a tuple $\mathbf{b} = (b_1, \ldots, b_s)$ of elements of J such that $\langle I, J \rangle \models \psi_{\mathbf{T}}(\mathbf{a}, \mathbf{b})$. Let h be a homomorphism from J to $\underline{\text{core}}(J)$. Then $h(a_i) = a_i$, since each a_i is a constant, for $1 \leq i \leq n$. Consequently, $\langle I, \underline{\text{core}}(J) \rangle \models \psi_{\mathbf{T}}(\mathbf{a}, h(\mathbf{b}))$, where $h(\mathbf{b}) = (h(b_1), \ldots, h(b_s))$. Thus, $\langle I, \underline{\text{core}}(J) \rangle$ satisfies the tgd.

Next, let $\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ be a tgd in Σ_t and $\mathbf{a} = (a_1, \ldots, a_n)$ a tuple of elements in $\underline{\text{core}}(J)$ such that $\underline{\text{core}}(J) \models \phi_{\mathbf{T}}(\mathbf{a})$. Since $\underline{\text{core}}(J)$ is a subinstance of J, and J is a solution, it follows that $J \models \phi_{\mathbf{T}}(\mathbf{a})$ and that there is a tuple $\mathbf{b} = (b_1, \ldots, b_s)$ of elements of J such that $J \models \psi_{\mathbf{T}}(\mathbf{a}, \mathbf{b})$. According to the last part of Proposition 4.2, there is a homomorphism h from J to $\underline{\text{core}}(J)$ such that $h(v) = v$, for every v in $\underline{\text{core}}(J)$. In particular, $h(a_i) = a_i$, for $1 \leq i \leq n$. It follows that $\underline{\text{core}}(J) \models \psi_{\mathbf{T}}(\mathbf{a}, h(\mathbf{b}))$, where $h(\mathbf{b}) = (h(b_1), \ldots, h(b_s))$. Thus, $\underline{\text{core}}(J)$ satisfies the tgd.

Finally, let $\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_1 = x_2)$ be an egd in Σ_t. If $\mathbf{a} = (a_1, \ldots, a_s)$ is a tuple of elements in $\underline{\text{core}}(J)$ such that $\underline{\text{core}}(J) \models \phi_{\mathbf{T}}(\mathbf{a})$, then $J \models \phi_{\mathbf{T}}(\mathbf{a})$, because $\underline{\text{core}}(J)$ is a subinstance of J. Since J is a solution, it follows that $a_1 = a_2$. Thus, $\underline{\text{core}}(J)$ satisfies every egd in Σ_t. ∎

COROLLARY 4.4. *Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting in which Σ_{st} is a set of tgds and Σ_t is the union of a set of tgds with a set of egds. If I is a source instance for which a universal solution exists, then there is a unique (up to isomorphism) universal solution J_0 for I having the following properties:*

- *J_0 is a core and is isomorphic to the core of every universal solution J for I.*
- *If J is a universal solution for I, there is a one-to-one homomorphism h from J_0 to J. Hence, $|J_0| \leq |J|$, where $|J_0|$ and $|J|$ are the sizes of J_0 and J.*

We refer to J_0 as *the core of the universal solutions (for I)*. As a very simple illustration of the concepts discussed in this subsection, recall the data exchange problem of Example 3.1. Then J_2 is the core of the universal solutions for I.

4.2 Query Answering with Cores

Corollary 4.4 reveals that, in addition to being unique, the core J_0 of the universal solutions for I is the *smallest* universal solution for I, and thus it is the most compact universal solution to materialize. In what follows, we show that using the core J_0 in query answering has clear advantages over using other universal solutions for this purpose. To begin with, if q is a union of conjunctive queries over the target schema, then $\mathit{certain}(q, I) = q(J_0)_{\downarrow}$, since J_0 is a universal solution. Suppose now that q is a conjunctive query with inequalities $\neq$ over the target schema. In general, if J is a universal solution, then $q(J)_{\downarrow}$ may properly contain $\mathit{certain}(q, I)$. In fact, this proper containment may hold even if J is the core J_0 of the universal solutions. Nonetheless, the next proposition shows that $q(J_0)_{\downarrow}$ equals the intersection of all choices of $q(J)$, where J varies over all universal solutions. Consequently, for unions of conjunctive queries with inequalities, query evaluation on the core of the universal solutions provides the best approximation (among all universal solutions) to the set of the certain answers. Moreover, the next proposition says that this property characterizes the core of the universal solutions.

PROPOSITION 4.5. *Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting in which Σ_{st} is a set of tgds and Σ_t is the union of a set of tgds with a set of egds. Let I be a source instance such that a universal solution for I exists, and let J_0 be the core of the universal solutions for I.*

1. *For every union q of conjunctive queries with inequalities $\neq$ on the target schema, $\mathit{certain}(q, I) \subseteq q(J_0)_{\downarrow} = \bigcap\{q(J) : J \text{ is a universal solution for } I\}$.*
2. *If J^* is a universal solution for I such that, for every conjunctive query q with inequalities $\neq$ on the target schema $\mathbf{T}$ and for every universal solution J, we have that $q(J^*)_{\downarrow} \subseteq q(J)$, then J^* is isomorphic to the core J_0 of the universal solutions.*

Proof: It is well known and easy to see that conjunctive queries with inequalities are preserved under one-to-one homomorphisms. That is, if h is a one-to-one homomorphism from J_1 to J_2, and if q is a k-ary conjunctive query with inequalities and $(a_1, \ldots, a_k) \in q(J_1)$, then $(h(a_1), \ldots, h(a_k)) \in q(J_2)$. Let J be a universal solution, and let J_0, J play the role of J_1, J_2 respectively (there is a one-to-one homomorphism h from the core J_0 to J by Corollary 4.4). It follows that if q is a k-ary conjunctive query with inequalities on the target schema $\mathbf{T}$ and $(a_1, \ldots, a_k)$ is a tuple of constants from I such that $(a_1, \ldots, a_k) \in q(J_0)$, then $(h(a_1), \ldots, h(a_k)) = (a_1, \ldots, a_k) \in q(J)$ (recall that the homomorphism h is the identity on the constants from I). Thus, $q(J_0)_{\downarrow} \subseteq q(J)$, and hence $q(J_0)_{\downarrow} \subseteq \bigcap\{q(J) : J \text{ universal for } I\}$. We now show the reverse inclusion. Define J_0' by renaming each null of J_0 in such a way that J_0 and J_0' have no nulls in common. Then $\bigcap\{q(J) : J \text{ universal for } I\} \subseteq q(J_0) \cap q(J_0')$. But it is easy to see that $q(J_0) \cap q(J_0') = q(J_0)_{\downarrow}$.

For the second part, assume that J^* is a universal solution for I such that for every union q of conjunctive queries with inequalities $\neq$ on the target schema $\mathbf{T}$ and for every universal solution J, we have that $q(J^*)_{\downarrow} \subseteq q(J)$. In particular, $q(J^*)_{\downarrow} \subseteq q(J_0)$. Let q^* be the *canonical* conjunctive query with inequalities associated with J^*, that is, q^* is a Boolean conjunctive query with inequalities that asserts that there exist n^* distinct elements, where n^* is the number of elements of J^*, and describes which tuples from J^* occur in which relations in the target schema $\mathbf{T}$. It is clear that $q^*(J^*) = \underline{\text{true}}$, which implies that $q^*(J_0) = \underline{\text{true}}$. In turn, $q^*(J_0) = \underline{\text{true}}$ implies that there is a one-to-one homomorphism h^* from J^* to J_0. At the same time, there is a one-to-one homomorphism from

J_0 to J^*, by Corollary 4.4. Hence J^* is isomorphic to J_0. ∎

Thus, null-free evaluation of a union q of conjunctive queries with inequalities on the core J_0 of the universal solutions gives precisely the set of all tuples that are guaranteed to appear in the output $q(J)$ of q on every universal solution J. The difference between the set $\bigcap\{\ q(J) : J$ is a universal solution for $I\}$ and the set *certain*(q, I) of the certain answers is that the latter is the intersection of the $q(J)$'s over all solutions J for I, not just the universal solutions for I. We have already argued that the universal solutions are the preferred solutions to the data exchange problem. Corollary 4.5 suggests that we could introduce an alternative notion of the certain answers based on universal solutions only. In this case, evaluating q on the core J_0 gives precisely the set of the certain answers according to this alternative notion.

Having established the good properties of the core of the universal solutions, we address next the problem of computing it. As mentioned earlier, universal solutions can be canonically computed by using the chase. However, the result of such a chase, while a universal solution, need not be the core. This section has established that, although it is well known that different chase sequences may yield non-isomorphic results, the universal solutions that are the results of the chase all have the same core. In the next two sections, we study what it takes to compute this core.

5. Complexity of Core Identification

Chandra and Merlin [4] were the first to realize that computing the core of a relational structure is an important problem in conjunctive query processing and optimization. Unfortunately, in its full generality this problem is intractable. Note that computing the core is a function problem, not a decision problem. One way to gauge the difficulty of a function problem is to analyze the computational complexity of its underlying decision problem.

DEFINITION 5.1. CORE IDENTIFICATION is the following decision problem: given two structures $\mathbf{A}$ and $\mathbf{B}$ over some schema $\mathbf{R}$ such that $\mathbf{B}$ is a substructure of $\mathbf{A}$, is $\underline{\mathsf{core}}(\mathbf{A}) = \mathbf{B}$? ∎

It is easy to see that CORE IDENTIFICATION is an NP-hard problem. Indeed, consider the following polynomial-time reduction from 3-COLORABILITY: a graph $\mathbf{G}$ is 3-colorable if and only if $\underline{\mathsf{core}}(\mathbf{G} \oplus \mathbf{K}_3) = \mathbf{K}_3$, where $\mathbf{K}_3$ is the complete graph with 3 nodes and $\oplus$ is the disjoint sum operation on graphs. This reduction was already given by Chandra and Merlin [4]. Later on, Hell and Nešetřil [12] studied the complexity of recognizing whether a graph is a core. In precise terms, CORE RECOGNITION is the following decision problem: given a structure $\mathbf{A}$ over some schema $\mathbf{R}$, is $\mathbf{A}$ a core? Clearly, this problem is in coNP. The main result in Hell and Nešetřil [12] asserts that CORE RECOGNITION is a coNP-complete problem, even if the inputs are undirected graphs. This is established by exhibiting a rather sophisticated polynomial-time reduction from NON-3-COLORABILITY on graphs of girth at least 7; the "gadgets" used in this reduction are pairwise incomparable cores with certain additional properties. It follows that CORE IDENTIFICATION is a coNP-hard problem. Nonetheless, it appears that the exact complexity of CORE IDENTIFICATION has not been pinpointed until now. In the sequel, we will establish that CORE IDENTIFICATION is a DP-complete problem. We present first some background material about the complexity class DP.

The class DP consists of all decision problems that can be written as the intersection of an NP-problem and a coNP-problem; equivalently, DP consists of all decision problems that can be written as the difference of two NP-problems. This class was introduced by Papadimitriou and Yannakakis [17], who discovered several DP-complete problems. The prototypical DP-complete problem is SAT/UNSAT: given two Boolean formulas ϕ and ψ, is ϕ satisfiable and ψ unsatisfiable? Several problems that express some "critical" property turn out to be DP-complete (see [18]). For instance, CRITICAL SAT is DP-complete, where an instance of this problem is a CNF-formula ϕ and the question is to determine whether ϕ is unsatisfiable, but if any one of its clauses is removed, then the resulting formula is satisfiable. Moreover, Cosmadakis [5] showed that certain problems related to database query evaluation are DP-complete. Note that DP contains both NP and coNP as subclasses; furthermore, each DP-complete problem is both NP-hard and coNP-hard. The prevailing belief in computational complexity is that the above containments are proper, but proving this remains an outstanding open problem. In any case, establishing that a certain problem is DP-complete is interpreted as signifying that this problem is intractable and, in fact, "more intractable" than an NP-complete problem.

Here, we establish that CORE IDENTIFICATION is a DP-complete problem by exhibiting a reduction from 3-COLORABILITY/NON-3-COLORABILITY on graphs of girth at least 7. This reduction is directly inspired by the reduction of NON-3-COLORABILITY on graphs of girth at least 7 to CORE RECOGNITION, given in [12].

THEOREM 5.2. CORE IDENTIFICATION *is DP-complete, even if the inputs are undirected graphs.*

In proving the above theorem, we make essential use of the following result, which is a special case of Theorem 6 in [12]. Recall that the *girth* of a graph is the length of the shortest cycle in the graph.

THEOREM 5.3. [12] *For each positive integer N, there is a sequence $\mathbf{A}_1, \ldots \mathbf{A}_N$ of connected graphs such that:*

1. *each $\mathbf{A}_i$ is 3-colorable, has girth 5, and each edge of $\mathbf{A}_i$ is on a 5-cycle;*
2. *each $\mathbf{A}_i$ is a core and, for every $i, j \leq n$ with $i \neq j$, there is no homomorphism from $\mathbf{A}_i$ to $\mathbf{A}_j$;*
3. *each $\mathbf{A}_i$ has at most $15(N + 4)$ nodes; and*
4. *there is a polynomial-time algorithm that, given N, constructs the sequence $\mathbf{A}_1, \ldots \mathbf{A}_N$.*

We now have the machinery needed to prove Theorem 5.2.

Proof of Theorem 5.2: CORE IDENTIFICATION is in DP, because, given two structures $\mathbf{A}$ and $\mathbf{B}$ over some schema $\mathbf{R}$ such that $\mathbf{B}$ is a substructure of $\mathbf{A}$, to determine whether $\underline{\mathsf{core}}(\mathbf{A}) = \mathbf{B}$ one has to check whether there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ (which is in NP) and whether $\mathbf{B}$ is a core (which is in coNP).

We will show that CORE IDENTIFICATION is DP-hard, even if the inputs are undirected graphs, via a polynomial-time reduction from 3-COLORABILITY/NON-3-COLORABILITY. As a stepping stone in this reduction, we will define CORE HOMOMORPHISM, which is the following variant of CORE IDENTIFICATION: given two structures $\mathbf{A}$ and $\mathbf{B}$, is there a homomorphism from $\mathbf{A}$ to $\mathbf{B}$,

and is $\mathbf{B}$ a core? There is a simple polynomial-time reduction of CORE HOMOMORPHISM to CORE IDENTIFICATION, where the instance $(\mathbf{A}, \mathbf{B})$ is mapped onto $(\mathbf{A} \oplus \mathbf{B}, \mathbf{B})$. This is a reduction, since there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ with $\mathbf{B}$ as a core if and only if $\underline{\mathsf{core}}(\mathbf{A} \oplus \mathbf{B}) = \mathbf{B}$. Thus, it remains to show that there is a polynomial-time reduction of 3-COLORABILITY/NON-3-COLORABILITY to CORE HOMOMORPHISM.

Hell and Nešetřil [12] showed that 3-COLORABILITY is NP-complete even if the input graphs have girth at least 7 (this follows from Theorem 7 in [12] by taking A to be a self-loop and B to be $\mathbf{K}_3$). Hence, 3-COLORABILITY/NON-3-COLORABILITY is DP-complete, even if the input graphs $\mathbf{G}$ and $\mathbf{H}$ have girth at least 7. So, assume that we are given two graphs $\mathbf{G}$ and $\mathbf{H}$ each having girth at least 7. Let $v_1, \ldots, v_m$ be an enumeration of the nodes of $\mathbf{G}$, let $w_1, \ldots, w_n$ be an enumeration of the nodes of $\mathbf{H}$, and let $N = m + n$. Let $\mathbf{A}_1, \ldots, \mathbf{A}_N$ be a sequence of connected graphs having the properties listed in Theorem 5.3. This sequence can be constructed in time polynomial in N; moreover, we can assume that these graphs have pairwise disjoint sets of nodes. Let $\mathbf{G}^*$ be the graph obtained by identifying each node v_i of $\mathbf{G}$ with some arbitrarily chosen node of $\mathbf{A}_i$, for $1 \leq i \leq m$ (and keeping the edges between nodes of $\mathbf{G}$ intact). Thus, the nodes of $\mathbf{G}^*$ are the nodes that appear in the $\mathbf{A}_i$'s, and the edges are the edges in the $\mathbf{A}_i$'s, along with the edges of $\mathbf{G}$ under our identification. Similarly, let $\mathbf{H}^*$ be the graph obtained by identifying each node w_j of $\mathbf{H}$ with some arbitrarily chosen node of $\mathbf{A}_j$, for $m+1 \leq j \leq N = m+n$ (and keeping the edges between nodes of $\mathbf{H}$ intact). We now claim that $\mathbf{G}$ is 3-colorable and $\mathbf{H}$ is not 3-colorable if and only if there is a homomorphism from $\mathbf{G}^* \oplus \mathbf{K}_3$ to $\mathbf{H}^* \oplus \mathbf{K}_3$, and $\mathbf{H}^* \oplus \mathbf{K}_3$ is a core. Hell and Nešetřil [12] showed that CORE RECOGNITION is coNP-complete by showing that a graph $\mathbf{H}$ of girth at least 7 is not 3-colorable if and only if the graph $\mathbf{H}^* \oplus \mathbf{K}_3$ is a core. We will use this property in order to establish the above claim.

Assume first that $\mathbf{G}$ is 3-colorable and $\mathbf{H}$ is not 3-colorable. Since each $\mathbf{A}_i$ is a 3-colorable graph, $\mathbf{G}^* \oplus \mathbf{K}_3$ is 3-colorable and so there a homomorphism from $\mathbf{G}^* \oplus \mathbf{K}_3$ to $\mathbf{H}^* \oplus \mathbf{K}_3$ (in fact, to $\mathbf{K}_3$). Moreover, as shown in [12], $\mathbf{H}^* \oplus \mathbf{K}_3$ is a core, since $\mathbf{H}$ is not 3-colorable. For the other direction, assume that there is a homomorphism from $\mathbf{G}^* \oplus \mathbf{K}_3$ to $\mathbf{H}^* \oplus \mathbf{K}_3$, and $\mathbf{H}^* \oplus \mathbf{K}_3$ is a core. Using again the results in [12], we infer that $\mathbf{H}$ is not 3-colorable. It remains to prove that $\mathbf{G}$ is 3-colorable. Let h be a homomorphism from $\mathbf{G}^* \oplus \mathbf{K}_3$ to $\mathbf{H}^* \oplus \mathbf{K}_3$. We claim that h actually maps $\mathbf{G}^*$ to $\mathbf{K}_3$; hence, $\mathbf{G}$ is 3-colorable. Let us consider the image of each graph $\mathbf{A}_i$, $1 \leq i \leq m$, under the homomorphism h. Observe that $\mathbf{A}_i$ cannot be mapped to some $\mathbf{A}_j$, $m+1 \leq j \leq N = m+n$, since, for every i and j such that $1 \leq i \leq m$ and $m+1 \leq j \leq N = m+n$, there is no homomorphism from $\mathbf{A}_i$ to $\mathbf{A}_j$. Observe also that the image of a cycle C under a homomorphism is a cycle C' of length less than or equal the length of C. Since $\mathbf{H}$ has girth at least 7 and since each edge of $\mathbf{A}_i$ is on a 5-cycle, the image of $\mathbf{A}_i$ under h cannot be contained in $\mathbf{H}$. For the same reason, the image of $\mathbf{A}_i$ under h cannot contain nodes from $\mathbf{H}$ and some $\mathbf{A}_j$, for $m+1 \leq j \leq N = m+n$; moreover, it cannot contain nodes from two different $\mathbf{A}_j$'s, for $m+1 \leq j \leq N = m+n$ (here, we also use the fact that each $\mathbf{A}_j$ has girth 5). Consequently, the homomorphism h must map each $\mathbf{A}_i$, $1 \leq i \leq m$, to $\mathbf{K}_3$. Hence, h maps $\mathbf{G}^*$ to $\mathbf{K}_3$, and so $\mathbf{G}$ is 3-colorable. ∎

We now consider the implications of the intractability of CORE RECOGNITION for the problem of computing the core of a structure. As stated earlier, Chandra and Merlin [4] observed that a graph $\mathbf{G}$ is 3-colorable if and only if $\underline{\mathsf{core}}(\mathbf{G} \oplus \mathbf{K}_3) = \mathbf{K}_3$. It follows that, unless P = NP, there is no polynomial-time algorithm for computing the core of a given structure. Indeed, if such an algorithm existed, then we could determine in polynomial-time whether a graph is 3-colorable by first running the algorithm to compute the core of $\mathbf{G} \oplus \mathbf{K}_3$ and then checking if the answer is equal to $\mathbf{K}_3$.

Note, however, that in data exchange we are interested in computing the core of a universal solution, rather than the core of an arbitrary instance. Consequently, we cannot assume a priori that the above intractability carries over to the data exchange setting, since polynomial-time algorithms for computing the core of universal solutions may exist. We address this next.

6. Computing the Core in Data Exchange

In contrast with the case of computing the core of an arbitrary instance, computing the core of a universal solution in data exchange does have polynomial-time algorithms, in certain natural data exchange settings. Section 6.1 gives a polynomial-time algorithm for computing the core of a universal solution, in a data exchange setting with no target constraints (i.e., $\Sigma_t = \emptyset$). We then show in Section 6.2 that essentially the same algorithm works (although the proof is quite a bit more complicated) if we remove the emptiness condition on Σ_t and allow it to contain egds.

6.1 A Polynomial-Time Case: No Target Constraints

We first define some notions that are needed in order to state the algorithm as well as to prove its correctness and polynomial-time bound. For the next two definitions, we assume K to be an arbitrary instance whose elements consists of constants from $\underline{\mathsf{Const}}$ and nulls from $\underline{\mathsf{Var}}$. We say that two elements of K are *adjacent* if there exists some tuple in some relation of K in which both elements occur.

DEFINITION 6.1. The *Gaifman graph of the nulls of* K is an undirected graph in which: 1) the nodes are all the nulls of K, and 2) there exists an edge between two nulls whenever the nulls are adjacent in K. A *block* of nulls is the set of nulls in a connected component of the Gaifman graph of nulls. ∎

If y is a null of K, then we may refer to the block of nulls that contains y as the *block of* y. Note that, by the definition of blocks, the set $\underline{\mathsf{Var}}(K)$ of all nulls of K is partitioned into disjoint blocks. Let K and K' be two instances with elements in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$. Recall that K' is a *subinstance* of K if every tuple of a relation of K' is a tuple of the corresponding relation of K.

DEFINITION 6.2. Let h be a homomorphism of K. Denote the result of applying h to K by $h(K)$. If $h(K)$ is a subinstance of K, then we call h an *endomorphism* of K. An endomorphism h of K is *useful* if $h(K) \neq K$ (i.e., $h(K)$ is a proper subinstance of K). ∎

The following lemma is a simple characterization of useful endomorphisms that we will make use of in proving the main results of this subsection and of subsection 6.2.

LEMMA 6.3. *Let K be an instance, and let h be an endomorphism of K. Then h is useful if and only if h is not one-to-one.*

Proof: Assume that h is not one-to-one. Then there is some x that is in the domain of h but not in the range of h (here we use the fact

that the instance is finite.) So no tuple containing x is in $h(K)$. Therefore, $h(K) \neq K$, and so h is useful.

Now assume that h is one-to-one. So h is simply a renaming of the members of K, and so an isomorphism of K. Thus, $h(K)$ has the same number of tuples as K. Since $h(K)$ is a subinstance of K, it follows that $h(K) = K$ (here again we use the fact that the instance K is finite). So h is not useful. ∎

For the rest of this subsection, we assume that we are given a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \emptyset)$ and a source instance I. Moreover, we assume that J is a canonical universal solution for this data exchange problem. That is, J is such that $\langle I, J\rangle$ is the result of chasing $\langle I, \emptyset\rangle$ with Σ_{st}. Our goal is to compute $\underline{\mathsf{core}}(J)$, that is, a subinstance C of J such that (1) $C = h(J)$ for some endomorphism h of J, and (2) there is no proper subinstance of C with the same property (condition (2) is equivalent to there being no endomorphism of C onto a proper subinstance of C). The central idea of the algorithm, as we shall see, is to show that the above mentioned endomorphism h of J can be found as the composition of a polynomial-length sequence of "local" (or "small") endomorphisms, each of which can be found in polynomial time. We next define what "local" means.

DEFINITION 6.4. Let K and K' be two instances such that the nulls of K' form a subset of the nulls of K, that is, $\underline{\mathsf{Var}}(K') \subseteq \underline{\mathsf{Var}}(K)$. Let h be some endomorphism of K', and let B be a block of nulls of K. We say that h is *K-local for B* if $h(x) = x$ whenever $x \notin B$. (Since all the nulls of K' are among the nulls of K, it makes sense to consider whether or not a null x of K' belongs to the block B of K.) We say that h is *K-local* if it is K-local for B, for some block B of K. ∎

The next lemma is crucial for the existence of the polynomial-time algorithm for computing the core of a universal solution.

LEMMA 6.5. *Assume a data exchange setting where Σ_{st} is a set of tgds and $\Sigma_t = \emptyset$. Let J' be a subinstance of the canonical universal solution J. If there exists a useful endomorphism of J', then there exists a useful J-local endomorphism of J'.*

Proof: Let h be a useful endomorphism of J'. By Lemma 6.3, we know that h is not one-to-one. So there is a null y that appears in J' but does not appear in $h(J')$. Let B be the block of y (in J). Define h' on J' by letting $h'(x) = h(x)$ if $x \in B$, and $h'(x) = x$ otherwise.

We show that h' is an endomorphism of J'. Let $(u_1, \ldots, u_s)$ be a tuple of the R relation of J'; we must show that $(h'(u_1), \ldots, h'(u_s))$ is a tuple of the R relation of J'. Since J' is a subinstance of J, the tuple $(u_1, \ldots, u_s)$ is also a tuple of the R relation of J. Hence, by definition of a block of J, all the nulls among $u_1, \ldots, u_s$ are in the same block B'. There are two cases, depending on whether or not $B' = B$. Assume first that $B' = B$. Then, by definition of h', for every u_i among $u_1, \ldots, u_s$, we have that $h'(u_i) = h(u_i)$ if u_i is a null, and $h'(u_i) = u_i = h(u_i)$ if u_i is a constant. Hence $(h'(u_1), \ldots, h'(u_s)) = (h(u_1), \ldots, h(u_s))$. Since h is an endomorphism of J', we know that $(h(u_1), \ldots, h(u_s))$ is a tuple of the R relation of J'. Thus, $(h'(u_1), \ldots, h'(u_s))$ is a tuple of the R relation of J'. Now assume that $B' \neq B$. So for every u_i among $u_1, \ldots, u_s$, we have that $h'(u_i) = u_i$. Hence $(h'(u_1), \ldots, h'(u_s)) = (u_1, \ldots, u_s)$. Therefore, once again, $(h'(u_1), \ldots, h'(u_s))$ is a tuple of the R relation of J', as desired. Hence, h' is an endomorphism of J'. ∎

We now present our algorithm for computing the core of the universal solutions, when $\Sigma_t = \emptyset$.

ALGORITHM 6.6. **(Core Algorithm)**
Input: source instance I.
Output: the core of the universal solutions for I.

1. Compute J, the canonical universal solution, from $\langle I, \emptyset\rangle$ by chasing with Σ_{st}.
2. Compute the blocks of J, and initialize J' to be J.
3. Check whether there exists a useful J-local endomorphism h of J'. If not, then stop with result J'.
4. Update J' to be $h(J')$, and return to Step 3.

In order to prove the upper bound on the execution time of the algorithm, we need to introduce two parameters. The first parameter, denoted by b, is the maximal number of existentially quantified variables over all tgds in Σ_{st}. Since we are taking Σ_{st} to be fixed, the quantity b is a constant. It follows easily from the construction of the canonical universal solution J (by chasing with Σ_{st}) that b is an upper bound on the size of a block in J. The second parameter, denoted by n, is the maximum of the number of elements and the number of tuples in the canonical universal solution J.

THEOREM 6.7. *Assume a data exchange setting where Σ_{st} is a set of tgds and $\Sigma_t = \emptyset$. Then Algorithm 6.6 is a correct, polynomial-time algorithm for computing the core of the universal solutions.*

Proof: We first show that Algorithm 6.6 is correct, that is, that the final instance C at the conclusion of the algorithm is the core of the given universal solution. Every time we apply Step 4 of the algorithm, we are replacing the instance by a homomorphic image. Therefore, the final instance C is the result of applying a composition of homomorphisms to the input instance, and hence is a homomorphic image of the canonical universal solution J. Also, since each of the homomorphisms found in Step 3 is an endomorphism, we have that C is a subinstance of J. Assume now that C is not the core; we shall derive a contradiction. Since C is not the core, there is an endomorphism h such that when h is applied to C, the resulting instance is a proper subinstance of C. Hence, h is a useful endomorphism of C. Therefore, by Lemma 6.5, there must exist a useful J-local endomorphism of C. But then Algorithm 6.6 should not have stopped in Step 3 with C. This is the desired contradiction. Hence, C is the core of J.

We now show that Algorithm 6.6 runs in polynomial time. Recall the parameters b and n introduced earlier: b is an upper bound on the size of a block of J, while n is the maximum of the total number of elements of J and the total number of tuples of J. Let J' be the instance in some execution of Step 3. For each block B, to check if there is a useful endomorphism of J' that is J-local for B, we can exhaustively check each of the possible functions h on the domain of J' such that $h(x) = x$ whenever $x \notin B$: there are at most n^b such functions. To check that such a function is actually a useful endomorphism requires time $O(n)$. Since there are at most n blocks, the time to determine if there is a block with a useful J-local endomorphism is $O(n^{b+2})$. The updating time in Step 4 is then $O(n)$.

By Lemma 6.3, after Step 4 is executed, there is at least one less null in J' than there was before. Since there are initially at most n nulls in the instance, it follows that the number of loops that Algorithm 6.6 performs is at most n. Therefore, the running time of the algorithm (except for Step 1 and Step 2, which are executed only once) is at most n (the number of loops) times $O(n^{b+2})$, that is, $O(n^{b+3})$. Since Step 1 and Step 2 take polynomial time as well, it follows that the entire algorithm executes in polynomial time. ∎

The crucial observation behind the polynomial-time bound is that the total number of endomorphisms that the algorithm explores in Step 3 is at most n^b for each block of J. This is in strong contrast with the case of minimizing arbitrary instances with constants and nulls for which we may need to explore a much larger number of endomorphisms (up to n^n, in general) in one minimization step.

6.2 Target Constraints

In this subsection, we extend Theorem 6.7 by showing that there is a polynomial-time algorithm for finding the core even when Σ_t is a set of egds. We do not know whether we can extend also to allow tgds, or even full tgds; this is an interesting open problem.

Thus, we assume next that we are given a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ where Σ_t is a set of egds. We are also given a source instance I. Let us denote by J the instance over the target schema such that $\langle I, J\rangle$ is the result of chasing $\langle I, \emptyset\rangle$ with Σ_{st}. Note that J is a canonical universal solution for the case when we replace Σ_t in the above data exchange setting with $\emptyset$. For this reason, let us call J the *canonical pre-universal instance*. Moreover, let us denote by J' the instance over the target schema such that J' is the result of chasing J with the set Σ_t of egds. That is, J' is a canonical universal solution for the given data exchange setting and source instance. Our goal is to compute $\underline{\mathsf{core}}(J')$, that is, a subinstance C of J' such that $C = h(J')$ for some endomorphism h of J', and such that there is no proper subinstance of C with the same property. As in the case when $\Sigma_t = \emptyset$, the central idea of the algorithm is to show that the above mentioned endomorphism h of J' can be found as the composition of a polynomial-length sequence of "small" endomorphisms, each findable in polynomial time. As in the case when $\Sigma_t = \emptyset$, "small" will mean J-local. We make this precise in the next lemma. This lemma, crucial for the existence of the polynomial-time algorithm for computing $\underline{\mathsf{core}}(J')$, is a non-trivial generalization of Lemma 6.5.

LEMMA 6.8. *Assume a data exchange setting where Σ_{st} is a set of tgds and Σ_t is a set of egds. Let J be the canonical pre-universal instance, and let J'' be an endomorphic image of the canonical universal solution J'. If there exists a useful endomorphism of J'', then there exists a useful J-local endomorphism of J''.*

The proof of Lemma 6.8 requires additional definitions as well as two additional lemmas. We start with the required definitions.

Let J be the canonical pre-universal instance, and let J' be the canonical universal solution produced from J by chasing with the set Σ_t of egds. We define a directed graph, whose nodes are the members of J, both nulls and constants. If during the chase process, a null u gets replaced by v (either a null or a constant), then there is an edge from u to v in the graph. Let $\leq$ be the reflexive, transitive closure of this graph. It is easy to see that $\leq$ is a reflexive partial order. For each node u, define $[u]$ to be the maximal (under $\leq$) node v such that $u \leq v$. Intuitively, u eventually gets replaced by $[u]$ as a result of the chase. It is clear that every member of J' is of the form $[u]$. It is also clear that if u is a constant, then $u = [u]$. Let us write $u \sim v$ if $[u] = [v]$. Intuitively, $u \sim v$ means that u and v eventually collapse to the same element as a result of the chase.

DEFINITION 6.9. Let K be an instance whose elements are constants and nulls. Let y be some element of K. We say that y is *rigid* if $h(y) = y$ for every homomorphism h of K. (In particular, all constants occuring in K are rigid.) ∎

A key step in the proof of Lemma 6.8 is the following surprising result, which says that if two nulls in different blocks of J both collapse onto the same element z of J' as a result of the chase, then z is *rigid*, that is, $h(z) = z$ for every endomorphism h of J'.

LEMMA 6.10 (RIGIDITY LEMMA). *Assume a data exchange setting where Σ_{st} is a set of tgds and Σ_t is a set of egds. Let J be the canonical pre-universal instance, and let J' be the result of chasing J with the set Σ_t of egds. Let x and y be nulls of J such that $x \sim y$, and such that $[x]$ is a non-rigid null of J'. Then x and y are in the same block of J.*

Proof: Assume that x and y are nulls in different blocks of J with $x \sim y$. We must show that $[x]$ is rigid in J'. Let ϕ be the diagram of the instance J, that is, the conjunction of all expressions $S(u_1, \ldots, u_s)$ where $(u_1, \ldots, u_s)$ is a tuple of the S relation of J. (We are treating members of J, both constants and nulls, as variables.) Let τ be the egd $\phi \rightarrow (x = y)$. Since $x \sim y$, it follows that $\Sigma_t \models \tau$. This is because the chase sets variables equal only when it is logically forced to (the result appears in papers that characterize the implication problem for dependencies; see, for instance, [3, 15]). Since J' satisfies Σ_t, it follows that J' satisfies τ.

We wish to show that $[x]$ is rigid in J'. Let h be a homomorphism of J'; we must show that $h([x]) = [x]$. Let B be the block of x in J. Let V be the assignment to the variables of τ obtained by letting $V(u) = h([u])$ if $u \in B$, and $V(u) = [u]$ otherwise. We now show that V is a valid assignment for ϕ in J', that is, that for each conjunct $S(u_1, \ldots, u_s)$ of ϕ, necessarily $(V(u_1), \ldots, V(u_s))$ is a tuple of the S relation of J'. Let $S(u_1, \ldots, u_s)$ be a conjunct of ϕ. By the construction of the chase, we know that $([u_1], \ldots, [u_s])$ is a tuple of the S relation of J', since $(u_1, \ldots, u_s)$ is a tuple of the S relation of J. There are two cases, depending on whether or not some u_i (with $1 \leq i \leq s$) is in B. If no u_i is in B, then $V(u_i) = [u_i]$ for each i, and so $(V(u_1), \ldots, V(u_s))$ is a tuple of the S relation of J', as desired. If some u_i is in B, then every u_i is either a null in B or a constant (this is because $(u_1, \ldots, u_s)$ is a tuple of the S relation of J). If u_i is a null in B, then $V(u_i) = h([u_i])$. If u_i is a constant, then $u_i = [u_i]$, and so $V(u_i) = [u_i] = u_i = h(u_i) = h([u_i])$, where the third equality holds since h is a homomorphism and u_i is a constant. Thus, in both cases, we have $V(u_i) = h([u_i])$. Since $([u_1], \ldots, [u_s])$ is a tuple of the S relation of J' and h is a homomorphism of J', we know that $(h[u_1], \ldots, h[u_s])$ is a tuple of the S relation of J'. So again, $(V(u_1), \ldots, V(u_s))$ is a tuple of the S relation of J', as desired.

Hence, V is a valid assignment for ϕ in J'. Therefore, since J' satisfies τ, it follows that in J', we have $V(x) = V(y)$. Now $V(x) = h([x])$, since $x \in B$. Further, $V(y) = [y]$, since $y \notin B$ (because y is in a different block than x). So $h([x]) = [y]$. Since $x \sim y$, that is, $[x] = [y]$, we have $h([x]) = [y] = [x]$, which shows that $h([x]) = [x]$, as desired. ∎

The contrapositive of Lemma 6.10 says that if x and y are nulls in different blocks of J that are set equal (perhaps transitively) during the chase, then $[x]$ is rigid in J'.

LEMMA 6.11. *Let h be an endomorphism of J'. Then every rigid element of J' is a rigid element of $h(J')$.*

Proof: Let u be a rigid element of J'. Then $h(u)$ is an element of $h(J')$, and so u is an element of $h(J')$, since $h(u) = u$ by rigidity. Let $\hat{h}$ be a homomorphism of $h(J')$; we must show that $\hat{h}(u) = u$. But $\hat{h}(u) = \hat{h}h(u)$, since $h(u) = u$. Now $\hat{h}h$ is also a homomorphism of J', since the composition of homomorphisms is a homomorphism. By rigidity of u in J', it follows that $\hat{h}h(u) = u$. So $\hat{h}(u) = \hat{h}h(u) = u$, as desired. ∎

We are now ready to give the proof of Lemma 6.8, after which we will present the minimization algorithm of this subsection.

Proof of Lemma 6.8: Let h be an endomorphism of J' such that $J'' = h(J')$, and let h' be a useful endomorphism of $h(J')$. By Lemma 6.3, there is a null y that appears in $h(J')$ but does not appear in $h'h(J')$. Let B be the block in J that contains y. Define h'' on $h(J')$ by letting $h''(x) = h'(x)$ if $x \in B$, and $h''(x) = x$ otherwise. We shall show that h'' is a useful J-local endomorphism of $h(J')$.

We now show that h'' is an endomorphism of $h(J')$. Let $(u_1, \ldots, u_s)$ be a tuple of the R relation of $h(J')$; we must show that $(h''(u_1), \ldots, h''(u_s))$ is a tuple of the R relation of $h(J')$.

We first show that every non-rigid null among $u_1, \ldots, u_s$ is in the same block of J. Let u_p and u_q be non-rigid nulls among $u_1, \ldots, u_s$; we show that u_p and u_q are in the same block of J. Since $(u_1, \ldots, u_s)$ is a tuple of the R relation of $h(J')$, and $h(J')$ is a subinstance of J', we know that $(u_1, \ldots, u_s)$ is a tuple of the R relation of J'. By construction of J' from J using the chase, we know that there is u_i' where $u_i \sim u_i'$ for $1 \leq i \leq s$, such that $(u_1', \ldots, u_s')$ is a tuple of the R relation of J. Since u_p and u_q are non-rigid nulls of $h(J')$, it follows from Lemma 6.11 that u_p and u_q are non-rigid nulls of J'. Now u_p' is not a constant, since $u_p' \sim u_p$ and u_p is a non-rigid null. Similarly, u_q' is not a constant. So u_p' and u_q' are in the same block B' of J. Now $[u_p] = u_p$, since u_p is in J'. Since $u_p' \sim u_p$ and $[u_p] = u_p$ is non-rigid, it follows from Lemma 6.10 that u_p' and u_p are in the same block of J, and so $u_p \in B'$. Similarly, $u_q \in B'$. So u_p and u_q are in the same block B' of J, as desired.

There are now two cases, depending on whether or not $B' = B$. Assume first that $B' = B$. For those u_i's that are non-rigid, we showed that $u_i \in B' = B$, and so $h''(u_i) = h'(u_i)$. For those u_j's that are rigid (including nulls and constants), we have $h''(u_j) = u_j = h'(u_j)$. So for every u_i among $u_1, \ldots, u_s$, we have $h''(u_j) = h'(u_j)$. Since h' is a homomorphism of $h(J')$, and since $(u_1, \ldots, u_s)$ is a tuple of the R relation of $h(J')$, we know that $(h'(u_1), \ldots, h'(u_s))$ is a tuple of the R relation of $h(J')$. Hence $(h''(u_1), \ldots, h''(u_s))$ is a tuple of the R relation of $h(J')$, as desired. Now assume that $B' \neq B$. For those u_i's that are non-rigid, we showed that $u_i \in B'$, and so $u_i \notin B$. Hence, for those u_i's that are non-rigid, we have $h''(u_j) = u_j$. But also $h''(u_i) = u_i$ for the rigid u_i's. Thus, $(h''(u_1), \ldots, h''(u_s)) = (u_1, \ldots, u_s)$. Hence, once again, $(h''(u_1), \ldots, h''(u_s))$ is a tuple of the R relation of $h(J')$, as desired.

So h'' is an endomorphism of $h(J')$. By definition, h'' is J-local. We show that h'' is useful. Since y appears in $h(J')$, Lemma 6.3 tells us that we need only show that the range of h'' does not contain y. If $x \in B$, then $h''(x) = h'(x) \neq y$, since the range of h' does not include y. If $x \notin B$, then $h''(x) = x \neq y$, since $y \in B$. So the range of h'' does not contain y, and hence h'' is useful. Therefore, h'' is a useful J-local endomorphism of $h(J')$. ∎

We now present our algorithm for computing the core when Σ_t is a set of egds. (As mentioned earlier, when the target constraints include egds, it may be possible that there are no solutions and hence no universal solutions. This case is detected by our algorithm, and "failure" is returned.)

ALGORITHM 6.12. **(egd-Core Algorithm)**
Input: source instance I.
Output: the core of the universal solutions for I, if solutions exist, and "failure", otherwise.

1. Compute J, the canonical pre-universal instance, from $\langle I, \emptyset \rangle$ by chasing with Σ_{st}.
2. Compute the blocks of J, and then chase J with Σ_t to produce the canonical universal solution J'. If the chase fails, then stop with "failure". Otherwise, initialize J'' to be J'.
3. Check whether there exists a useful J-local endomorphism h of J''. If not, then stop with result J''.
4. Update J'' to be $h(J'')$, and return to Step 3.

THEOREM 6.13. *Assume a data exchange setting where Σ_{st} is a set of tgds and Σ_t is a set of egds. Then Algorithm 6.12 is a correct, polynomial-time algorithm for computing the core of the universal solutions.*

Proof: The proof is essentially the same as that of Theorem 6.7, except that we make use of Lemma 6.8 instead of Lemma 6.5. We also use the fact that chasing with egds (used in Step 2) is a polynomial-time procedure. ∎

We note that it is essential for the polynomial-time upper bound that the endomorphisms explored by Algorithm 6.12 are J-local and not merely J'-local. While, as argued earlier in the case $\Sigma_t = \emptyset$, the blocks of J are bounded in size by the constant b (the maximal number of existentially quantified variables over all tgds in Σ_{st}), the same is not true, in general, for the blocks of J'. The chase with egds, used to obtain J', may generate blocks of unbounded size. Intuitively, if an egd equates the nulls x and y that are in different blocks of J, then this creates a new, larger, block out of the union of the blocks of x and y.

7. Conclusions and Open Problems

In a previous paper [9], we argued that universal solutions are the best solutions in a data exchange setting, in that they are the "most general possible" solutions. Unfortunately, there may be many universal solutions. In this paper, we pick a particular universal solution, namely, the core of an arbitrary universal solution, and argue that it is the best universal solution (and hence the best of the best). The core is unique up to isomorphism, and is the universal solution of the smallest size (that is, with the fewest tuples). The core gives the best answer, among all universal solutions, for queries that are the union of conjunctive queries with inequalities. By "best

answer", we mean that the core provides the best approximation (among all universal solutions) to the set of the certain answers. In fact, if we redefine the set of "certain answers" to be those that occur in every universal solution, then the core gives the exact answer.

We then consider the question of the complexity of computing the core. We show that the complexity of deciding if a graph H is the core of a graph G is DP-complete. Unless P = NP, there is no polynomial-time algorithm for producing the core of a given arbitrary structure. On the other hand, in our case of interest, namely data exchange, we give natural conditions where there are polynomial-time algorithms for computing the core of universal solutions.

We believe that this paper opens the door to a number of fascinating questions. First, there are questions about the performance of the core in query-answering. How well does the core perform in answering queries more general then unions of conjunctive queries with inequalities? Second, there are questions about the complexity of constructing the core. Even in the case where we prove that there is a polynomial-time algorithm for computing the core (when the set Σ_t of target constraints contains only egds), the exponent may be somewhat large. Is there a more efficient algorithm for computing the core in this case? There is also the question of extending the polynomial-time result more generally. Is there a polynomial-time algorithm for computing the core when Σ_t consists of full tgds? And what about a very general data exchange setting where Σ_t contains an arbitrary set of egds and (not-necessarily-full) tgds? On a slightly different note, and given the similarities between the two problems, it would be interesting to see if our techniques for minimizing universal solutions can be applied to the problem of minimizing the chase-generated universal plans that arise in the comprehensive query optimization method introduced in [7].

Finally, the work reported here addresses data exchange only between relational schemas. In the future we hope to investigate to what extent the results presented in this paper and in [9] can be extended to the more general case of XML/nested data exchange.

Acknowledgments. Many thanks to Renée J. Miller, Val Tannen and Moshe Y. Vardi for helpful suggestions.

8. REFERENCES

[1] S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *PODS*, pages 254–263, 1998.

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

[3] C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

[4] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*, pages 77–90, 1977.

[5] S. Cosmadakis. The Complexity of Evaluating Relational Queries. *Information and Control*, 58:101–112, 1983.

[6] S. S. Cosmadakis and P. C. Kanellakis. Functional and Inclusion Dependencies: A Graph Theoretic Approach. In *Advances in Computing Research*, volume 3, pages 163–184. JAI Press, 1986.

[7] A. Deutsch, L. Popa, and V. Tannen. Physical Data Independence, Constraints and Optimization with Universal Plans. In *VLDB*, pages 459–470, 1999.

[8] A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints. In *ICDT*, pages 225–241, 2003.

[9] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *ICDT*, pages 207–224, 2003.

[10] M. Friedman, A. Y. Levy, and T. D. Millstein. Navigational Plans For Data Integration. In *AAAI*, pages 67–73, 1999.

[11] A. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, pages 270–294, 2001.

[12] P. Hell and J. Nešetřil. The Core of a Graph. *Discrete Mathematics*, 109:117–126, 1992.

[13] P. C. Kanellakis. Elements of Relational Database Theory. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 1073–1156. Elsevier and MIT Press, 1990.

[14] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[15] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing Implications of Data Dependencies. *ACM TODS*, 4(4):455–469, Dec. 1979.

[16] R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *VLDB*, pages 77–88, 2000.

[17] C. Papadimitriou and M. Yannakakis. The Complexity of Facets and Some Facets of Complexity. In *STOC*, pages 229–234, 1982.

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

[19] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.

[20] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. *ACM TODS*, 2(2):134–174, 1977.

[21] R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.