Data Exchange: Getting to the Core

RONALD FAGIN
IBM Almaden Research Center
fagin@almaden.ibm.com
and
PHOKION G. KOLAITIS[1]
IBM Almaden Research Center
kolaitis@almaden.ibm.com
and
LUCIAN POPA
IBM Almaden Research Center
lucian@almaden.ibm.com

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, and 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. We investigate the computational complexity of producing the core. Well-known results by Chandra and Merlin imply that, unless P = 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. We also 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. Finally, we show that the core is the best among all universal solutions for answering existential queries, and we propose an alternative semantics for answering queries in data exchange settings.

[1]On leave from UC Santa Cruz; partially supported by NSF Grant IIS-9907419.

A preliminary version of this paper appeared in Proc. 2003 ACM Symposium of Principles of Database Systems, San Diego, pp. 90–101.

Categories and Subject Descriptors: H.2.5 [**Heterogeneous Databases**]: Data translation; H.2.4 [**Systems**]: Relational Databases; H.2.4 [**Systems**]: Query processing

General Terms: data exchange, data integration, computational complexity, query answering
Additional Key Words and Phrases: certain answers, conjunctive queries, core, universal solutions, dependencies, chase

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 [Shu et al. 1977] 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 under-specification 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 [Fagin et al. 2003], 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 development of Clio [Miller et al. 2000; Popa et al. 2002] at the IBM Almaden Research Center. Clio is a prototype system for schema mapping and data exchange between autonomous applications. A data exchange setting is a quadruple ($\mathbf{S}$, $\mathbf{T}$, Σ_{st}, Σ_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 [Fagin et al. 2003], 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 [van der Meyden 1998] for a survey). Moreover, the certain answers have been used for query answering in data integration [Lenzerini 2002]. In the same data integration context, Abiteboul and Duschka [Abiteboul and Duschka 1998] studied the complexity of computing the certain answers.

We showed [Fagin et al. 2003] 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 [Beeri and Vardi 1984; Maier et al. 1979].

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, [Hell and Nešetřil 1992]). Moreover, in the early days of database theory, Chandra and Merlin [Chandra and Merlin 1977] 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 [Abiteboul et al. 1995; Chandra and Merlin 1977; Kanellakis 1990]).

Consider a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ in which Σ_{st} is a set of source-to-target tgds and Σ_t is a set of target tgds and target 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.

After this, we address the issue of how hard it is to compute the core of a universal solution. Chandra and Merlin [Chandra and Merlin 1977] 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 are polynomial-time algorithms for computing the core of universal solutions in data exchange settings in which Σ_{st} is a set of source-to-target tgds and Σ_t is a set of target egds. It remains an open problem to determine whether this result can be extended to data exchange settings in which the target constraints Σ_t consist of both egds and tgds. 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 [Chandra and Merlin 1977] imply that this problem is NP-hard. Later on, Hell and Nešetřil [Hell and Nešetřil 1992] 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.

In the last part of the paper, we further justify the selection of the core as the "best" universal solution by establishing its usefulness in answering queries over the target schema $\mathbf{T}$. An *existential query* $q(\mathbf{x})$ is a formula of the form $\exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$, where $\phi(\mathbf{x}, \mathbf{y})$ is a quantifier-free formula.[2] Perhaps the most important examples of existential queries are the conjunctive queries with inequalities $\neq$. Another useful example of existential queries is the set-difference query, which asks whether there is a member of the set difference $A - B$.

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 [Fagin et al. 2003], however, it was shown that there are simple conjunctive queries with inequalities $\neq$ such that evaluating them on a universal solution always produces a *proper superset* of the set of certain answers for I. Nonetheless, here we show that evaluating existential queries on the core J_0 of the universal solutions yields the *best approximation* (that is, the smallest superset) of the set of the certain answers, among all universal solutions. Analogous to the definition of certain answers, let us define *the certain answers on universal solutions* of a query q over the target schema, with respect to a source instance I, to be the set of all tuples that are in the intersection of all $q(J)$'s, as J varies over all *universal* solutions for I; we write

[2] We shall also give a safety condition on ϕ.

u-certain(q, I) to denote the certain answers of q on universal solutions for I. Since we consider universal solutions to be the preferred solutions to the data exchange problem, this suggests the naturalness of this notion of certain answers on universal solutions as an alternative semantics for query answering in data exchange settings. We show that if q is an existential query and J_0 is the core of the universal solutions for I, then the set of those tuples in $q(J_0)$ whose entries are elements from the source instance I is equal to the set u-certain(q, I) of the certain answers of q on universal solutions. We also show that in the LAV setting (an important scenario in data integration) there is an interesting contrast between the complexity of computing certain answers and of computing certain answers on universal solutions. Specifically, Abiteboul and Duschka [Abiteboul and Duschka 1998] showed that there is a data exchange setting with $\Sigma_t = \emptyset$ and a conjunctive query with inequalities $\neq$ such that computing the certain answers of this query is a coNP-complete problem. In contrast to this, we establish here that in an even more general data exchange setting ($\mathbf{S}$, $\mathbf{T}$, Σ_{st}, Σ_t) in which Σ_{st} is an arbitrary set of tgds and Σ_t is an arbitrary set of egds, for every existential query q (and in particular, for every conjunctive query q with inequalities $\neq$), there is a polynomial-time algorithm for computing the set u-certain(q, I) of the certain answers of q on universal solutions.

2 Preliminaries

This section contains the main definitions related to data exchange and a minimum amount of background material. The presentation follows closely our earlier paper [Fagin et al. 2003].

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*. An instance I can be identified with the set of all facts arising from the relations R_i^I of I. 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 $x_1, \ldots, x_k$. We assume that all the variables in $\mathbf{x}$ appear free in $\phi_{\mathbf{S}}(\mathbf{x})$. 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 [Popa et al. 2002], 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 (all results except for Proposition 2.7), each source-to-target dependency in Σ_{st} is a *tuple generating dependency (tgd)* [Beeri and Vardi 1984] 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 $\mathbf{T}$. We assume that all the variables in $\mathbf{x}$ appear in $\phi_{\mathbf{S}}(\mathbf{x})$. 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)* [Beeri and Vardi 1984], 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 $\mathbf{T}$, where all the variables in $\mathbf{x}$ appear in $\phi_{\mathbf{T}}(\mathbf{x})$, and x_1, x_2 are among the variables in $\mathbf{x}$. The tgds and egds together comprise Fagin's (embedded) implicational dependencies [Fagin 1982]. As in [Fagin et al. 2003], 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 [Popa et al. 2002]. 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 [Lenzerini 2002] for a detailed discussion of LAV and GAV data integration systems and sound views).

A *LAV data integration system with sound views defined by conjunctive queries* is a special case of a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, 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 [Friedman et al. 1999; Lenzerini 2002]. 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. Consider a data exchange problem in which the source schema consists of two binary relation symbols as follows: `EmpCity`, associating employees with cities they work in, and `LivesIn`, associating employees with cities they live in. Assume that the target schema consists of three binary relation symbols as follows: `Home`, associating employees with their home cities, `EmpDept`, associating employees with departments, and `DeptCity`, associating departments with their cities. We assume that $\Sigma_t = \emptyset$. The source-to-target tgds and the source instance are as follows, where (d_1), (d_2), (d_3), and (d_4) are labels for convenient reference later:

$$\begin{aligned}
\Sigma_{st}:\ &(d_1)\ \texttt{EmpCity}(e,c) \rightarrow \exists H\, \texttt{Home}(e,H)\\
&(d_2)\ \texttt{EmpCity}(e,c) \rightarrow \exists D(\texttt{EmpDept}(e,D) \wedge \texttt{DeptCity}(D,c))\\
&(d_3)\ \texttt{LivesIn}(e,h) \rightarrow \texttt{Home}(e,h)\\
&(d_4)\ \texttt{LivesIn}(e,h) \rightarrow \exists D \exists C(\texttt{EmpDept}(e,D) \wedge \texttt{DeptCity}(D,C))
\end{aligned}$$

$$\begin{aligned}
I = \{\ &\texttt{EmpCity}(\text{Alice}, \text{SJ}), \texttt{EmpCity}(\text{Bob}, \text{SD}),\\
&\texttt{LivesIn}(\text{Alice}, \text{SF}), \texttt{LivesIn}(\text{Bob}, \text{LA})\ \}
\end{aligned}$$

We shall use this example as a running example throughout this paper. 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:

$$\begin{aligned}
J_0 = \{\ &\texttt{Home}(\text{Alice}, \text{SF}), \texttt{Home}(\text{Bob}, \text{SD}),\\
&\texttt{EmpDept}(\text{Alice}, D_1), \texttt{EmpDept}(\text{Bob}, D_2),\\
&\texttt{DeptCity}(D_1, \text{SJ}), \texttt{DeptCity}(D_2, \text{SD})\ \},
\end{aligned}$$

where D_1 and D_2 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 occurring 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 [van der Meyden 1998]) as well as in the context of data integration (see [Halevy 2001; Lenzerini 2002]).

Intuitively, in the above instance, D_1 and D_2 are used to "give values" for the existentially quantified variable D of (d_2), in order to satisfy (d_2) for the two source tuples $\texttt{EmpCity}(\text{Alice}, \text{SJ})$ and $\texttt{EmpCity}(\text{Bob}, \text{SD})$. In contrast, two constants (SF and SD) are used to "give values" for the existentially quantified variable H of (d_1), in order to satisfy (d_1) for the same two source tuples.

The following instances are solutions as well:

$$J = \{ \quad \texttt{Home}(\text{Alice}, \text{SF}), \texttt{Home}(\text{Bob}, \text{SD}), \\ \texttt{Home}(\text{Alice}, H_1), \texttt{Home}(\text{Bob}, H_2), \\ \texttt{EmpDept}(\text{Alice}, D_1), \texttt{EmpDept}(\text{Bob}, D_2), \\ \texttt{DeptCity}(D_1, \text{SJ}), \texttt{DeptCity}(D_2, \text{SD}) \},$$

$$J'_0 = \{ \quad \texttt{Home}(\text{Alice}, \text{SF}), \texttt{Home}(\text{Bob}, \text{SD}), \\ \texttt{EmpDept}(\text{Alice}, D), \texttt{EmpDept}(\text{Bob}, D), \\ \texttt{DeptCity}(D, \text{SJ}), \texttt{DeptCity}(D, \text{SD}) \},$$

The instance J differs from J_0 by having two extra Home tuples where the home cities of Alice and Bob are two nulls, H_1 and H_2, respectively. The second instance J'_0 differs from J_0 by using the same null (namely D) to denote the "unknown" department of both Alice and Bob. ∎

Next, we review the notion of universal solutions, proposed in [Fagin et al. 2003] 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 instance J'_0 in Example 2.2 is not universal. In particular, there is no homomorphism from J'_0 to J_0. Hence, the solution J'_0 contains "extra" information that was not required by the specification; in particular, J'_0 "assumes" that the departments of Alice and Bob are the same. In contrast, it can easily be shown that J_0 and J have homomorphisms to every solution (and to each other). Thus, J_0 and J are universal solutions. ∎

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 [Fagin et al. 2003], 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. *[Fagin et al. 2003] 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 [Deutsch and Tannen 2003] and in [Fagin et al. 2003] (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 [Beeri and Vardi 1984] and sets of acyclic inclusion dependencies [Cosmadakis and Kanellakis 1986]. 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 [Fagin et al. 2003]) 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 In a data exchange setting, there may be many different solutions for a given source instance. Hence, given a source instance, the question arises as to what the result of answering queries over the target schema is. Following earlier work on information integration, in [Fagin et al. 2003] we adopted the notion of the certain answers as the semantics of query answering in data exchange settings. As stated in Section 1, 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; in symbols,

$$\textit{certain}(q, I) \;=\; \bigcap \{q(J) : J \text{ is a solution for } I\}.$$

Before stating the connection between the certain answers and universal solutions, let us recall the definitions of conjunctive queries (with inequalities) and unions of conjunctive queries (with inequalities). A *conjunctive query* $q(\mathbf{x})$ over a schema $\mathbf{R}$ is a formula of the form $\exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$ where $\phi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathbf{R}$. If, in addition to atomic formulas, the conjunction $\phi(\mathbf{x}, \mathbf{y})$ is allowed to contain inequalities of the form $z_i \neq z_j$, where z_i, z_j are variables among $\mathbf{x}$ and $\mathbf{y}$, we call $q(\mathbf{x})$ a *conjunctive*

query with inequalities. We also impose a safety condition, that every variable in $\mathbf{x}$ and $\mathbf{y}$ must appear in an atomic formula, not just in an inequality. A *union of conjunctive queries (with inequalities)* is a disjunction $q(\mathbf{x}) = q_1(\mathbf{x}) \vee \ldots \vee q_n(\mathbf{x})$ where $q_1(\mathbf{x}), \ldots, q_n(\mathbf{x})$ are conjunctive queries (with inequalities).

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 [Fagin et al. 2003] asserts 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. *[Fagin et al. 2003] 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 Data Exchange with Cores

3.1 Multiple Universal Solutions

Even if we restrict attention to universal solutions instead of arbitrary solutions, there may still exist multiple, non-isomorphic universal solutions for a given instance of a data exchange problem. Moreover, although these universal solutions are homomorphically equivalent to each other, they may have different sizes (where the *size* is the number of tuples). The following example illustrates this state of affairs.

EXAMPLE 3.1. We again revisit our running example from Example 2.2. As we noted earlier, of the three target instances given there, two of them (namely, J_0 and J) are universal solutions for I. These are nonisomorphic universal solutions (since they have different sizes). We now give an infinite family of nonisomorphic universal solutions, that we shall make use of later.

For every $m \geq 0$, let J_m be the target instance

$$\begin{aligned} J_m = \{ \quad & \texttt{Home}(\text{Alice}, \text{SF}), \texttt{Home}(\text{Bob}, \text{SD}), \\ & \texttt{EmpDept}(\text{Alice}, X_0), \texttt{EmpDept}(\text{Bob}, Y_0), \\ & \texttt{DeptCity}(X_0, \text{SJ}), \texttt{DeptCity}(Y_0, \text{SD}), \\ & \ldots \\ & \texttt{EmpDept}(\text{Alice}, X_m), \texttt{EmpDept}(\text{Bob}, Y_m), \\ & \texttt{DeptCity}(X_m, \text{SJ}), \texttt{DeptCity}(Y_m, \text{SD}) \ \}, \end{aligned}$$

where $X_0, Y_0, \ldots, X_m, Y_m$ are distinct labeled nulls. (In the case of $m = 0$, the resulting instance J_0 is the same, modulo renaming of nulls, as the earlier J_0 from Example 2.2. We take the liberty of using the same name, since the choice of nulls really does not matter.) It is easy to verify that each target instance J_m, for $m \geq 0$, is a universal solution for I; thus, there are infinitely many non-isomorphic universal solutions for I. It is also easy to see that every universal solution must contain at least four tuples $\texttt{EmpDept}(\text{Alice}, X)$,

$\texttt{EmpDept}(\text{Bob}, Y)$, $\texttt{DeptCity}(X, \text{SJ})$, and $\texttt{DeptCity}(Y, \text{SD})$, for some labeled nulls X and Y, as well as the tuples $\texttt{Home}(\text{Alice}, \text{SF})$ and $\texttt{Home}(\text{Bob}, \text{SD})$. Consequently, the instance J_0 has the smallest size among all universal solutions for I and actually is the unique (up to isomorphism) universal solution of smallest size. Thus, J_0 is a rather special universal solution and, from a size point of view, a preferred candidate to materialize in data exchange. ∎

Motivated by the preceding example, in the sequel we introduce and study the concept of the *core* of a universal solution. We show that the core of a universal solution is the unique (up to isomorphism) smallest universal solution. We then address the problem of computing the core and also investigate the use of cores in answering queries over the target schemas. The results that we will establish make a compelling case that cores are the preferred solutions to materialize in data exchange.

3.2 Cores and Universal Solutions

In addition to the notion of an *instance* over a schema (which we defined earlier), we find it convenient to define the closely related notion of a *structure* over a schema. 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 each $R_i^{\mathbf{A}}$ is a relation on A of the same arity as the relation symbol 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. A structure $\mathbf{B} = (B, R_1^{\mathbf{B}}, \ldots, R_k^{\mathbf{B}})$ is an *induced substructure* of $\mathbf{A}$ if $B \subseteq A$ and, for every $1 \leq i \leq k$, we have that $R_i^{\mathbf{B}} = \{(x_1, \ldots, x_n) \mid R_i^{\mathbf{A}}(x_1, \ldots, x_n)$ and $x_1, \ldots, x_n$ are in $B\}$.

DEFINITION 3.2. 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 [Hell and Nešetřil 1992]). The next proposition summarizes some basic facts about cores; a proof can be found in [Hell and Nešetřil 1992].

PROPOSITION 3.3. *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* $\mathbf{C}$*.*
- *If* $\mathbf{C}$ *is the core of a finite structure* $\mathbf{A}$*, then* $\mathbf{C}$ *is an induced substructure of* $\mathbf{A}$*.*

In view of Proposition 3.3, 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.

Universal solutions for I are unique up to homomorphic equivalence, but as we saw in Example 3.1, they need not be unique up to isomorphism. Proposition 3.3, however, implies that their cores are isomorphic; in other words, all universal solutions for I have the same core up to isomorphism. Moreover, if J is a universal solution for I and $\underline{\mathsf{core}}(J)$ is a solution for I, then $\underline{\mathsf{core}}(J)$ is also a universal solution for I, since J and $\underline{\mathsf{core}}(J)$ 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 a set of tgds and egds.

PROPOSITION 3.4. *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 a set of tgds and egds. If* I *is a source instance and* J *is a solution for* I*, then* $\underline{\mathsf{core}}(J)$ *is a solution for* I*. Consequently, if* J *is a universal solution for* I*, then also* $\underline{\mathsf{core}}(J)$ *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 $\underline{\mathsf{core}}(J)$. Then $h(a_i) = a_i$, since each a_i is a constant, for $1 \leq i \leq n$. Consequently, $\langle I, \underline{\mathsf{core}}(J) \rangle \models \psi_{\mathbf{T}}(\mathbf{a}, h(\mathbf{b}))$, where $h(\mathbf{b}) = (h(b_1), \ldots, h(b_s))$. Thus, $\langle I, \underline{\mathsf{core}}(J) \rangle$ 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 $\underline{\mathsf{core}}(J)$ such that $\underline{\mathsf{core}}(J) \models \phi_{\mathbf{T}}(\mathbf{a})$. Since $\underline{\mathsf{core}}(J)$ is a subinstance of J, it follows that $J \models \phi_{\mathbf{T}}(\mathbf{a})$, and since J is a solution, it follows 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 3.3, there is a homomorphism h from J to $\underline{\mathsf{core}}(J)$ such that $h(v) = v$, for every v in $\underline{\mathsf{core}}(J)$. In particular, $h(a_i) = a_i$, for $1 \leq i \leq n$. It follows that $\underline{\mathsf{core}}(J) \models \psi_{\mathbf{T}}(\mathbf{a}, h(\mathbf{b}))$, where $h(\mathbf{b}) = (h(b_1), \ldots, h(b_s))$. Thus, $\underline{\mathsf{core}}(J)$ 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 $\underline{\mathsf{core}}(J)$ such that $\underline{\mathsf{core}}(J) \models \phi_{\mathbf{T}}(\mathbf{a})$, then $J \models \phi_{\mathbf{T}}(\mathbf{a})$, because $\underline{\mathsf{core}}(J)$ is a subinstance of J. Since J is a solution, it follows that $a_1 = a_2$. Thus, $\underline{\mathsf{core}}(J)$ satisfies every egd in Σ_t. ∎

COROLLARY 3.5. *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 a set of tgds and 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 an illustration of the concepts discussed in this subsection, recall the data exchange problem of Example 3.1. Then J_0 is indeed the core of the universal solutions for I.

The core of the universal solutions is the preferred universal solution to materialize in data exchange, since it is the unique most compact universal solution. In turn, this raises the question of how to compute cores of universal solutions. 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. In general, an algorithm other than the chase is needed for computing cores of universal solutions. In the next two sections, we study what it takes to compute cores. We begin by analyzing the complexity of computing cores of arbitrary instances and then focus on the computation of cores of universal solutions in data exchange.

4 Complexity of Core Identification

Chandra and Merlin [Chandra and Merlin 1977] 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 4.1. 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 [Chandra and Merlin 1977]. Later on, Hell and Nešetřil [Hell and Nešetřil 1992] 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. Hell and Nešetřil's main result [Hell and Nešetřil 1992] is 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 in the literature 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 [Papadimitriou and Yannakakis 1982], 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 [Papadimitriou 1994]). 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 [Cosmadakis 1983] 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 [Hell and Nešetřil 1992].

THEOREM 4.2. 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 [Hell and Nešetřil 1992]. Recall that the *girth* of a graph is the length of the shortest cycle in the graph.

THEOREM 4.3. [Hell and Nešetřil 1992] *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; moreover, for every i, j with $i \leq n$, $j \leq n$ and $i \neq j$, 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 4.2.

Proof of Theorem 4.2: 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 [Hell and Nešetřil 1992] showed that 3-COLORABILITY is NP-complete even if the input graphs have girth at least 7 (this follows from Theorem 7 in [Hell and Nešetřil 1992] 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 4.3. 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 [Hell and Nešetřil 1992] 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 [Hell and Nešetřil 1992], $\mathbf{H}^* \oplus \mathbf{K}_3$ 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 [Hell and Nešetřil 1992], 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 $\mathbf{A}_i$, with $1 \leq i \leq m$, under the homomorphism h. Observe that $\mathbf{A}_i$ cannot be mapped to some $\mathbf{A}_j$, when $m + 1 \leq j \leq N = m + n$, 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. ∎

It should be noted that problems equivalent to CORE RECOGNITION and CORE IDENTIFICATION have been investigated in logic programming and artificial intelligence. Specifically, Gottlob and Fermüller [Gottlob and Fermüller 1993] studied the problem of removing redundant literals from a clause, and analyzed the computational complexity of two related decision problems: the problem of determining whether a given clause is *condensed* and the problem of determining whether, given two clauses, one is a *condensation* of the other. Gottlob and Fermüller showed that the first problem is coNP-complete and the

second is DP-complete. As it turns out, determining whether a given clause is condensed is equivalent to CORE RECOGNITION, while determining whether a clause is a condensation of another clause is equivalent to CORE IDENTIFICATION. Thus, the complexity of CORE RECOGNITION and CORE IDENTIFICATION for relational structures (but not for undirected graphs) can also be derived from the results in [Gottlob and Fermüller 1993]. As a matter of fact, the reductions in [Gottlob and Fermüller 1993] give easier proofs for the coNP-hardness and DP-hardness of CORE RECOGNITION and CORE IDENTIFICATION respectively for undirected graphs *with constants*, that is, undirected graphs in which certain nodes are distinguished so that every homomorphism maps each such constant to itself (alternatively, graphs with constants can be viewed as relational structures with a binary relation for the edges and unary relations each of which consists of one of the constants). For instance, the coNP-hardness of CORE IDENTIFICATION for graphs with constants can be established via the following reduction from the CLIQUE problem. Given an undirected graph $\mathbf{G}$ and a positive integer k, consider the disjoint sum $\mathbf{G} \oplus \mathbf{K}_k$, where $\mathbf{K}_k$ is the complete graph with k elements. If every node in $\mathbf{G}$ is viewed as a constant, then $\mathbf{G} \oplus \mathbf{K}_k$ is a core if and only if $\mathbf{G}$ does not contain a clique with k elements.

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 [Chandra and Merlin 1977] 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 $\mathrm{P} = \mathrm{NP}$, 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.

5 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. Specifically, in this section we show that the core of a universal solution can be computed in polynomial time in data exchange settings in which Σ_{st} is an arbitrary set of tgds and Σ_t is a set of egds.

We give two rather different polynomial-time algorithms for the task of computing the core in data exchange settings in which Σ_{st} is an arbitrary set of tgds and Σ_t is a set of egds: a *greedy* algorithm and an algorithm we call the *blocks* algorithm. Section 5.1 is devoted to the greedy algorithm. In Section 5.2 we present the blocks algorithm for data exchange settings with no target constraints (i.e., $\Sigma_t = \emptyset$). We then show in Section 5.3 that essentially the same blocks algorithm works if we remove the emptiness condition on Σ_t and allow it to contain egds. Although the blocks algorithm is more complicated than the greedy algorithm (and its proof of correctness much more involved), it has certain advantages for data exchange that we will describe later on.

In what follows, we assume that $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ is a data exchange setting such that Σ_{st}

is a set of tgds and Σ_t is a set of egds. Given a source instance I, we let J be the target instance obtained by chasing $\langle I, \emptyset \rangle$ with Σ_{st}. We call J a *canonical pre-universal instance* for I. Note that J is a canonical universal solution for I with respect to the data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \emptyset)$ (that is, no target constraints).

5.1 Greedy Algorithm

Intuitively, given a source instance I, the greedy algorithm first determines whether solutions for I exist, and then, if solutions exist, computes the core of the universal solutions for I by successively removing tuples from a canonical universal solution for I, as long as I and the instance resulting in each step satisfy the tgds in Σ_{st}. Recall that a *fact* is an expression of the form $R(t)$ indicating that the tuple t belongs to the relation R; moreover, every instance can be identified with the set of all facts arising from the relations of that instance.

ALGORITHM 5.1. **Greedy Algorithm**
Input: source instance I.
Output: the core of the universal solutions for I, if solutions exist; "failure", otherwise.

1. Chase I with Σ_{st} to produce a canonical pre-universal instance J.
2. Chase J with Σ_t; if the chase fails, then stop and return "failure"; otherwise, let J' be the canonical universal solution for I produced by the chase.
3. Initialize J^* to be J'.
4. While there is a fact $R(t)$ in J^* such that $\langle I, J^* - \{R(t)\} \rangle$ satisfies Σ_{st}, set J^* to be $J^* - \{R(t)\}$.
5. Return J^*

THEOREM 5.2. *Assume that $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ is a data exchange setting such that Σ_{st} is a set of tgds and Σ_t is a set of egds. Then Algorithm 5.1 is a correct, polynomial-time algorithm for computing the core of universal solutions.*

Proof: As shown in [Fagin et al. 2003] (see also Theorem 2.6), the chase is a correct, polynomial-time algorithm for determining whether, given a source instance I, a solution exists and, if so, producing the canonical universal solution J'.

Assume that for a given source instance I, a canonical universal solution J' for I has been produced in Step 2 of the greedy algorithm. We claim that each target instance J^* produced during the iterations of the while loop in Step 4 is a universal solution for I. To begin with, $\langle I, J^* \rangle$ satisfies the tgds in Σ_{st} by construction. Furthermore, J^* satisfies the egds in Σ_t, because J^* is a subinstance of J', and J' satisfies the egds in Σ_t. Consequently, J^* is a solution for I; moreover, it is a universal solution, since it is a subinstance of the canonical universal solution J for I and thus it can be mapped homomorphically into every solution for I.

Let C be the target instance returned by the algorithm. Then C is a universal solution for I and hence it contains an isomorphic copy J_0 of the core of the universal solutions as a subinstance. We claim that $C = J_0$. Indeed, if there is a fact $R(t)$ in $C - J_0$, then $C - \{R(t)\}$ satisfies the tgds in Σ_{st}, since J_0 satisfies the tgds in Σ_{st} and it is a subinstance of $J_0 - \{R(t)\}$; thus, the algorithm could not have returned C as output.

In order to analyze the running time of the algorithm, we consider the following parameters: m is the size of the source instance I (number of tuples in I); a is the maximum

number of universally quantified variables over all tgds in Σ_{st}; b is the maximum number of existentially quantified variables over all tgds in Σ_{st}; finally, a' is the maximum number of universally quantified variables over all egds in Σ_t. Since the data exchange setting is fixed, the quantities a, b, and a' are constants.

Given a source instance I of size m, the size of the canonical pre-universal instance J is $O(m^a)$ and the time needed to produce it is $O(m^{a+ab})$. Indeed, the canonical pre-universal instance is constructed by considering each tgd $(\forall \mathbf{x})(\varphi_{\mathbf{S}}(\mathbf{x}) \rightarrow (\exists \mathbf{y})\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$ in Σ_{st}, instantiating the universally quantified variables $\mathbf{x}$ with elements from I in every possible way, and, for each such instantiation, checking whether the existentially quantified variables $\mathbf{y}$ can be instantiated by existing elements so that the formula $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ is satisfied, and, if not, adding null values and facts to satisfy it. Since Σ_{st} is fixed, at most a constant number of facts are added at each step, which accounts for the $O(m^a)$ bound in the size of the canonical pre-universal instance. There are $O(m^a)$ possible instantiations of the universally quantified variables, and for each such instantiation $O((m^a)^b)$ steps are needed to check whether the existentially quantified variables can be instantiated by existing elements, hence the total time required to construct the canonical pre-universal instance is $O(m^{a+ab})$.

The size of the canonical universal solution J' is also $O(m^a)$ (since it is at most the size of J) and the time needed to produce J' from J is $O(m^{aa'+2a})$. Indeed, chasing with the egds in Σ_t requires at most $O((m^a)^2) = O(m^{2a})$ chase steps, since in the worst case every two values will be set equal to each other. Moreover, each chase step takes time $O((m^a)^{a'})$, since at each step we need to instantiate the universally quantified variables in the egds in every possible way.

The while loop in Step 4 requires at most $O(m^a)$ iterations each of which takes $O(m^{a+ab})$ steps to verify that Σst is satisfied by $\langle I, J^* - \{R(t)\}\rangle$. Thus, Step 4 takes time $O(m^{2a+ab})$. It follows that the the running time of the greedy algorithm is $O(m^{2a+ab} + m^{2a+aa'})$. ∎

Several remarks are in order now. First, it should be noted that the correctness of the greedy algorithm depends crucially on the assumption that Σ_t consists of egds only. The crucial property that holds for egds, but fails for tgds, is that if an instance satisfies an egd, then every subinstance of it also satisfies that egd. Thus, if the greedy algorithm is applied to data exchange settings in which Σ_t contains at least one tgd, then the output of the algorithm may fail to be a solution for the input instance. One can consider a variant of the greedy algorithm in which the test in the while loop is that $\langle I, J^* - \{R(t)\}\rangle$ satisfies both Σ_{st} and Σ_t. This modified greedy algorithm outputs a universal solution for I, but it is not too hard to construct examples in which the output is not the core of the universal solutions for I.

Note that Step 4 of the greedy algorithm can also be construed as a polynomial-time algorithm for producing the core of the universal solutions, given a source instance I and some arbitrary universal solution J' for I. The first two steps of the greedy algorithm produce a universal solution for I in time polynomial in the size of the source instance I or determine that no solution for I exists, so that the entire greedy algorithm runs in time polynomial in the size of I.

Although the greedy algorithm is conceptually simple and its proof of correctness transparent, it requires that the source instance I be available throughout the execution of the algorithm. There are situations, however, in which the original source I becomes unavailable, after a canonical universal solution J' for I has been produced. In particular, the

Clio system [Popa et al. 2002] uses a specialized engine to produce a canonical universal solution, when there are no target constraints, or a canonical pre-universal instance, when there are target constraints. Any further processing, such as chasing with target egds or producing the core, will have to be done by another engine or application that may not have access to the original source instance.

This state of affairs raises the question of whether the core of the universal solutions can be produced in polynomial time using only a canonical universal solution or only a canonical pre-universal instance. In what follows, we describe such an algorithm, called the *blocks algorithm*, which has the feature that it can start from either a canonical universal solution or a canonical pre-universal instance, and has no further need for the source instance. We present the blocks algorithms in two stages: first, for the case in which there are no target constraints ($\Sigma_t = \emptyset$), and then for the case in which Σ_t is a set of egds.

5.2 Blocks Algorithm: No Target Constrains

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 5.3. 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 the 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 5.4. 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 5.3.

LEMMA 5.5. *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 5.6. 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 5.7. *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 5.5, 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 5.8. **Blocks Algorithm: No Target Constraints**

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.

THEOREM 5.9. *Assume that $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ is a data exchange setting such that Σ_{st} is a set of tgds and $\Sigma_t = \emptyset$. Then Algorithm 5.8 is a correct, polynomial-time algorithm for computing the core of the universal solutions.*

Proof: We first show that Algorithm 5.8 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 5.7, there must exist a useful J-local endomorphism of C. But then Algorithm 5.8 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 5.8 runs in polynomial time. To do so, we need to consider certain parameters. As in the analysis of greedy algorithm, the first parameter, denoted by b, is the maximum 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 size of the canonical universal solution J (number of tuples in J); as seen in the analysis of the greedy algorithm, n is $O(m^a)$, where a is the maximum number of the universally quantified variables over all tgds in Σ_{st} and m is the size of I. 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 $O(n)$.

By Lemma 5.5, 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 5.8 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.

5.3 Blocks Algorithm: Target egds

In this subsection, we extend Theorem 5.9 by showing that there is a polynomial-time algorithm for finding the core even when Σ_t is a set of egds.

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. As with the greedy algorithm, let J be a canonical pre-universal instance, that is, J is the result of chasing I with Σ_{st}. Let J' be the canonical universal solution obtained by chasing J with Σ_t. 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 5.7.

LEMMA 5.10. *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 5.10 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 5.11. 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 occurring in K are rigid.) ∎

A key step in the proof of Lemma 5.10 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 5.12 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, [Beeri and Vardi 1984; Maier et al. 1979]). 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 5.12 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 5.13. *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 5.10, after which we will present the blocks algorithm for the case of target egds.

Proof of Lemma 5.10: Let h be an endomorphism of J' such that $J'' = h(J')$, and let h' be a useful endomorphism of $h(J')$. By Lemma 5.5, 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 5.13 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 5.12 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 now show that h'' is useful. Since y appears in $h(J')$, Lemma 5.5 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 5.14. **Blocks Algorithm: Target egds**
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 5.15. *Assume that $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ is a data exchange setting such that Σ_{st} is a set of tgds and Σ_t is a set of egds. Then Algorithm 5.14 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 5.9, except that we make use of Lemma 5.10 instead of Lemma 5.7. For the correctness of the algorithm, we use the fact that each $h(J'')$ is both a homomorphic image and a subinstance of the canonical universal solution J', hence it satisfies both the tgds in Σ_{st} and the egds in Σ_t. For the running time of the algorithm, 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 5.14 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.

5.4 Can We Obtain the Core Via the Chase?

A universal solution can be obtained via the chase [Fagin et al. 2003]. What about the core? In this section, we show by example that the core may not be obtainable via the chase. We begin with a preliminary example.

EXAMPLE 5.16. We again consider our running example from Example 2.2. If we chase the source instance I of Example 2.2 by first chasing with the dependencies (d_2) and (d_3), and then by the dependencies (d_1) and (d_4), neither of which add any tuples, then the result is the core J_0, as given in Example 2.2. If, however, we chase first with the dependency (d_1), then with the dependencies (d_2) and (d_3), and finally with the dependency (d_4), which does not add any tuples, then the result is the target instance J, as given in Example 2.2, rather than the core J_0. ∎

In Example 5.16 , the result of the chase may or may not be the core, depending on the order of the chase steps. We now give an example where there is no chase (that is, no order of doing the chase steps) that produces the core.

EXAMPLE 5.17. Assume that the source schema consists of one 4-ary relation symbol R and the target schema consists of one 5-ary relation symbol S. There are two source-to-target tgds d_1 and d_2, where d_1 is

$$R(a,b,c,d) \rightarrow \exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 (\begin{array}{rl} S(x_5,b,x_1,x_2,a) & \wedge \\ S(x_5,c,x_3,x_4,a) & \wedge \\ S(d,c,x_3,x_4,b)) & \end{array}$$

and where d_2 is

$$R(a,b,c,d) \rightarrow \exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 (\begin{array}{rl} S(d,a,a,x_1,b) & \wedge \\ S(x_5,a,a,x_1,a) & \wedge \\ S(x_5,c,x_2,x_3,x_4)) & \end{array}$$

The source instance I is $\{R(1,1,2,3)\}$.

The result of chasing I with d_1 only is

$$\{ \begin{array}{l} S(N_5,1,N_1,N_2,1), \\ S(N_5,2,N_3,N_4,1), \end{array} \quad (1)$$

$$S(3, 2, N_3, N_4, 1) \quad \}$$

where N_1, N_2, N_3, N_4, N_5 are nulls.

The result of chasing I with d_2 only is

$$\begin{array}{ll} \{ & S(3, 1, 1, N_1', 1), \\ & S(N_5', 1, 1, N_1', 1), \\ & S(N_5', 2, N_2', N_3', N_4') \ \} \end{array} \tag{2}$$

where $N_1', N_2', N_3', N_4', N_5'$ are nulls.

Let J be the universal solution that is the union of (1) and (2). We now show that the core of J is given by the following instance J_0, which consists of the third tuple of (1) and the first tuple of (2):

$$\begin{array}{ll} \{ & S(3, 2, N_3, N_4, 1), \\ & S(3, 1, 1, N_1', 1) \quad \}. \end{array}$$

First, it is straightforward to verify that J_0 is the image of the universal solution J under the following endomorphism h: $h(N_1) = 1$; $h(N_2) = N_1'$; $h(N_3) = N_3$; $h(N_4) = N_4$; $h(N_5) = 3$; $h(N_1') = N_1'$; $h(N_2') = N_3$; $h(N_3') = N_4$; $h(N_4') = 1$; and $h(N_5') = 3$. Second, it is easy to see that there is no endomorphism of J_0 into a proper substructure of J_0. From these two facts, it follows immediately that J_0 is the core.

Since the result of chasing first with d_1 has three tuples, and since the core has only two tuples, it follows that the result of chasing first with d_1 and then d_2 does not give the core. Similarly, the result of chasing first with d_2 and then d_1 does not give the core. Thus, no chase gives the core, which was to be shown.

This example has several other features built into it. First, it is not possible to remove a conjunct from the right-hand side of d_1 and still maintain a dependency equivalent to d_1. A similar comment applies to d_2. Therefore, the fact that no chase gives the core is not caused by the right-hand side of a source-to-target tgd having a redundant conjunct.

Second, the Gaifman graph of the nulls as determined by (1) is connected. Intuitively, this tells us that the tgd d_1 cannot be "decomposed" into multiple tgds with the same left-hand side. A similar comment applies to d_2. Therefore, the fact that no chase gives the core is not caused by the tgds being "decomposable".

Third, not only does the set (1) of tuples not appear in the core, but even the core of (1), which consists of the first and third tuples of (1), does not appear in the core. A similar comment applies to (2), whose core consists of the first and third tuples of (2). So even if we were to modify the chase by inserting, at each chase step, only the core of the set of tuples generated by applying a given tgd, we still would not obtain the core as the result of a chase. ∎

6 Query Answering with Cores

Up to this point, we have shown that there are two reasons for using cores in data exchange: first, they are the smallest universal solutions, and second, they are polynomial-time computable in many natural data exchange settings. In this section, we provide further justification for using cores in data exchange by establishing that they have clear advantages over other universal solutions in answering target queries.

Assume that $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ is a data exchange setting, I is a source instance, and J_0 is the core of the universal solutions for I. If q is a union of conjunctive queries over the target schema $\mathbf{T}$, then, by Proposition 2.7, for every universal solution J for I, we have that *certain*$(q, I) = q(J)_{\downarrow}$. In particular, *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 *certain*(q, I). We illustrate this point with the following example.

EXAMPLE 6.1. Let us revisit our running example from Example 2.2. We saw earlier in Example 3.1 that for every $m \geq 0$, the target instance

$$\begin{aligned} J_m = \{ \quad & \texttt{Home}(\text{Alice}, \text{SF}), \texttt{Home}(\text{Bob}, \text{SD}), \\ & \texttt{EmpDept}(\text{Alice}, X_0), \texttt{EmpDept}(\text{Bob}, Y_0), \\ & \texttt{DeptCity}(X_0, \text{SJ}), \texttt{DeptCity}(Y_0, \text{SD}), \\ & \quad \ldots \\ & \texttt{EmpDept}(\text{Alice}, X_m), \texttt{EmpDept}(\text{Bob}, Y_m), \\ & \texttt{DeptCity}(X_m, \text{SJ}), \texttt{DeptCity}(Y_m, \text{SD}) \ \} \end{aligned}$$

is a universal solution for I; moreover, J_0 is the core of the universal solutions for I. Consider now the following conjunctive query with one inequality:

$$q(e) \ = \ \exists D_1 \exists D_2 \ (\texttt{EmpDept}(e, D_1) \wedge \texttt{EmpDept}(e, D_2) \wedge (D_1 \neq D_2)).$$

Clearly, $q(J_0) = \emptyset$, while if $m \geq 1$, then $q(J_m) = \{\text{Alice}, \text{Bob}\}$. This implies that *certain*$(q, I) = \emptyset$, and thus evaluating the above query q on the universal solution J_m, for arbitrary $m \geq 1$, produces a strict superset of the set of the certain answers. In contrast, evaluating q on the core J_0 coincides with the set of the certain answers, since $q(J_0) = \emptyset =$ *certain*(q, I).

This example can also be used to illustrate another difference between conjunctive queries and conjunctive queries with inequalities. Specifically, if J and J' are universal solutions for I, and q^* is a conjunctive query over the target schema, then $q^*(J)_{\downarrow} = q^*(J')_{\downarrow}$. In contrast, this does not hold for the above conjunctive query q with one inequality. Indeed, $q(J_0) = \emptyset$ while $q(J_m) = \{\text{Alice}, \text{Bob}\}$, for every $m \geq 1$. ∎

In the preceding example, the certain answers of a particular conjunctive query with inequalities could be obtained by evaluating the query on the core of the universal solutions. As shown in the next example, however, this does not hold true for arbitrary conjunctive queries with inequalities.

EXAMPLE 6.2. Referring to our running example, consider again the universal solutions J_m, for $m \geq 0$, from Example 6.1. In particular, recall the instance J_0, which is the core of the universal solutions for I, and which has two distinct labeled nulls X_0 and Y_0, denoting unknown departments. Besides their role as placeholders for department values, the role of such nulls is also to "link" employees to the cities they work in, as specified by the tgd (d_2) in Σ_{st}. For data exchange, it is important that such nulls be different from constants and different from each other. Universal solutions such as J_0 naturally satisfy this requirement. In contrast, the target instance

$$\begin{aligned} J_0' = \{ \quad & \texttt{Home}(\text{Alice}, \text{SF}), \texttt{Home}(\text{Bob}, \text{SD}), \\ & \texttt{EmpDept}(\text{Alice}, X_0), \texttt{EmpDept}(\text{Bob}, X_0), \\ & \texttt{DeptCity}(X_0, \text{SJ}), \texttt{DeptCity}(X_0, \text{SD}) \ \}, \end{aligned}$$

is a solution[3] for I, but not a universal solution for I, because it uses the same null for both source tuples $(\text{Alice}, \text{SJ})$ and (Bob, SD) and hence, there is no homomorphism from J_0' to J_0. In this solution, the association between Alice and SJ as well as the association between Bob and SD have been lost.

Let $q(e, c)$ be the following conjunctive query with one inequality:

$$\exists D \exists D' \, (\texttt{EmpDept}(e, D) \wedge \texttt{DeptCity}(D', c) \wedge (D \neq D')).$$

It is easy to see that $q(J_0) = \{(\text{Alice}, \text{SD}), (\text{Bob}, \text{SJ})\}$. In contrast, $q(J_0') = \emptyset$, since in J_0' both Alice and Bob are linked with both SJ and SD. Consequently, *certain*$(q, I) = \emptyset$, and thus *certain*(q, I) is properly contained in $q(J_0)_{\downarrow}$.

Let J be a universal solution for I. Since J_0 is (up to a renaming of the nulls) the core of J, it follows that

$$q(J_0) \subseteq q(J)_{\downarrow}.$$

(We are using the fact that $q(J_0) = q(J_0)_{\downarrow}$ here.) Since also we have the strict inclusion *certain*$(q, I) \subset q(J_0)$, we have that *certain*$(q, I) \subset q(J)_{\downarrow}$, for every universal solution J. This also means that there is no universal solution J for I such that *certain*$(q, I) = q(J)_{\downarrow}$.

Finally, consider the target instance:

$$\begin{aligned} J' = \{ \quad & \texttt{Home}(\text{Alice}, \text{SF}), \texttt{Home}(\text{Bob}, \text{SD}), \\ & \texttt{EmpDept}(\text{Alice}, X_0), \texttt{EmpDept}(\text{Bob}, Y_0), \\ & \texttt{DeptCity}(X_0, \text{SJ}), \texttt{DeptCity}(Y_0, \text{SD}), \\ & \texttt{DeptCity}(X', \text{SJ}) \; \} \end{aligned}$$

It is easy to verify that J' is a universal solution and that $q(J') = \{ \, (\text{Alice}, \text{SJ}), (\text{Alice}, \text{SD}), (\text{Bob}, \text{SJ}) \, \}$. Thus, the following strict inclusions hold: *certain*$(q, I) \subset q(J_0)_{\downarrow} \subset q(J')_{\downarrow}$. This shows that a strict inclusion hierarchy can exist among the set of the certain answers, the result of the null-free query evaluation on the core and the result of the null-free query evaluation on some other universal solution.

We will argue in the next section that instead of computing *certain*(q, I) a better answer to the query may be given by taking $q(J_0)_{\downarrow}$ itself! ∎

6.1 Certain Answers on Universal Solutions

Although the certain answers of conjunctive queries with inequalities cannot always be obtained by evaluating these queries on the core of the universal solutions, it turns out that this evaluation produces a "best approximation" to the certain answers among all evaluations on universal solutions. Moreover, as we shall show, this property characterizes the core, and also extends to existential queries.

We now define existential queries, including a safety condition. An *existential query* $q(\mathbf{x})$ is a formula of the form $\exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$, where $\phi(\mathbf{x}, \mathbf{y})$ is a quantifier-free formula in disjunctive normal form. Let ϕ be $\vee_i \wedge_j \gamma_{ij}$, where each γ_{ij} is an atomic formula, the negation of an atomic formula, an equality, or the negation of an equality. As a safety condition, we assume that for each conjunction $\wedge_j \gamma_{ij}$ and each variable z (in $\mathbf{x}$ or $\mathbf{y}$) that appears in this conjunction, one of the conjuncts γ_{ij} is an atomic formula that contains z.

[3] This is the same instance, modulo renaming of nulls, as the earlier instance J_0' of Example 2.2.

The safety condition guarantees that ϕ is *domain independent* [Fagin 1982] (so that its truth does not depend on any underlying domain, but only on the "active domain" of elements that appear in tuples in the instance).

We now introduce the following concept, which we shall argue is fundamental.

DEFINITION 6.3. Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting and let I be a source instance. For every query q over the target schema $\mathbf{T}$, the set of *the certain answers of q on universal solutions with respect to the source instance I*, denoted by u-certain(q, I), is the set of all tuples that appear in $q(J)$ for every universal solution J for I; in symbols,

$$\text{u-certain}(q, I) \;=\; \bigcap\{q(J) : J \text{ is a universal solution for } I\}.$$

Clearly, *certain*$(q, I) \subseteq$ u-certain(q, I). Moreover, if q is a union of conjunctive queries, then Proposition 2.7 implies that *certain*$(q, I) =$ u-certain(q, I). In contrast, if q is a conjunctive query with inequalities, it is possible that *certain*(q, I) is properly contained in u-certain(q, I). Concretely, this holds true for the query q and the source instance I in Example 6.2, since *certain*$(q, I) = \emptyset$, while u-certain$(q, I) = \{(\text{Alice}, \text{SD}), (\text{Bob}, \text{SJ})\}$. In such cases, there is no universal solution J for I such that *certain*$(q, I) = q(J)_{\downarrow}$. Nonetheless, the next result asserts that if J_0 is the core of the universal solutions for I, then u-certain$(q, I) = q(J_0)_{\downarrow}$. Therefore, $q(J_0)_{\downarrow}$ is the best approximation (that is, the least superset) of the certain answers for I among all choices of $q(J)_{\downarrow}$ where J is a universal solution for I.

Before we prove the next result, we need to recall some definitions from [Fagin et al. 2003]. Let q be a Boolean (that is, 0-ary) query over the target schema $\mathbf{T}$ and I a source instance. If we let $\underline{\mathsf{true}}$ denote the set with one 0-ary tuple and $\underline{\mathsf{false}}$ denote the empty set, then each of the statements $q(J) = \underline{\mathsf{true}}$ and $q(J) = \underline{\mathsf{false}}$ has its usual meaning for Boolean queries q. It follows from the definitions that *certain*$(q, I) = \underline{\mathsf{true}}$ means that for every solution J of this instance of the data exchange problem, we have that $q(J) = \underline{\mathsf{true}}$; moreover, *certain*$(q, I) = \underline{\mathsf{false}}$ means that there is a solution J such that $q(J) = \underline{\mathsf{false}}$.

PROPOSITION 6.4. *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 a set of tgds and 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*) *If q is an existential query over the target schema $\mathbf{T}$, then*

$$\text{u-certain}(q, I) \;=\; q(J_0)_{\downarrow}.$$

(*2*) *If J^* is a universal solution for I such that for every existential query q over the target schema T we have that*

$$\text{u-certain}(q, I) \;=\; q(J^*)_{\downarrow},$$

then J^* is isomorphic to the core J_0 of the universal solutions for I. In fact, it is enough for the above property to hold for every conjunctive query q with inequalities $\neq$.

Proof: Let J be a universal solution, and let J_0 be the core of J. By Proposition 3.3, we know that J_0 is an induced substructure of J. Let q be an existential query over the target schema $\mathbf{T}$. Since q is an existential query and J_0 is an induced substructure of J, it is straightforward to verify that $q(J_0) \subseteq q(J)$ (this is a well-known preservation property

of existential first-order formulas). Since J_0 is the core of every universal solution for I up to a renaming of the nulls, it follows that $q(J_0)_{\downarrow} \subseteq \bigcap\{q(J) : J \text{ universal for } I\}$. 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 $q(J_0) \cap q(J_0') = q(J_0)_{\downarrow}$. This proves the reverse inclusion and so

$$\text{u-certain}(q, I) = \bigcap\{q(J) : J \text{ universal for } I\} = q(J_0)_{\downarrow}.$$

For the second part, assume that J^* is a universal solution for I such that for every conjunctive query q with inequalities $\neq$ over the target schema,

$$q(J^*)_{\downarrow} = \bigcap\{q(J) : J \text{ is a universal solution for } I\}. \tag{3}$$

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 at least 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 $q^*(J^*) = \underline{\mathsf{true}}$. Since q^* is a Boolean query, we have $q(J^*)_{\downarrow} = q(J^*)$. So from (3), where q^* plays the role of q, we have

$$q^*(J^*) = \bigcap\{q^*(J) : J \text{ is a universal solution for } I\}. \tag{4}$$

Since $q^*(J^*) = \underline{\mathsf{true}}$, it follows from (4) that $q^*(J_0) = \underline{\mathsf{true}}$. In turn, $q^*(J_0) = \underline{\mathsf{true}}$ 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 3.5. Consequently, J^* is isomorphic to J_0. ∎

Let us take a closer look at the concept of the certain answers of a query q on universal solutions. In [Fagin et al. 2003], we made a case that the universal solutions are the preferred solutions to the data exchange problem, since in a precise sense they are the most general possible solutions and, thus, they represent the space of all solutions. This suggests that, in the context of data exchange, the notion of the certain answers on universal solutions may be more fundamental and more meaningful than that of the certain answers. In other words, we propose here that u-certain(q, I) should be used as the semantics of query answering in data exchange settings, instead of *certain*(q, I), because we believe that this notion should be viewed as the "right" semantics for query answering in data exchange. As pointed out earlier, *certain*(q, I) and u-certain(q, I) coincide when q is a union of conjunctive queries, but they may very well be different when q is a conjunctive query with inequalities. The preceding Example 6.2 illustrates this difference between the two semantics, since *certain*$(q, I) = \emptyset$ and u-certain$(q, I) = \{(\text{Alice}, \text{SD}), (\text{Bob}, \text{SJ})\}$, where $q(e, c)$ is the query

$$\exists D \exists D' \, (\texttt{EmpDept}(e, D) \wedge \texttt{DeptCity}(D', c) \wedge (D \neq D')).$$

We argue that a user should not expect the empty set $\emptyset$ as the answer to the query q, after the data exchange between the source of the target (unless, of course, further constraints are added to specify that the nulls must be equal). Thus, u-certain$(q, I) = \{(\text{Alice}, \text{SD}), (\text{Bob}, \text{SJ})\}$ is a more intuitive answer to q than *certain*$(q, I) = \emptyset$. Furthermore, this answer can be computed as $q(J_0)_{\downarrow}$.

We now show that for conjunctive queries with inequalities, it may be easier to compute the certain answers on universal solutions than to compute the certain answers. Abiteboul

and Duschka proved the following result.

THEOREM 6.5. [Abiteboul and Duschka 1998] *There is a* LAV *setting and a Boolean conjunctive query q with inequalities $\neq$ such that computing the set certain(q, I) of the certain answers of q is a* coNP*-complete problem.*

By contrast, we prove the following result, which covers not only LAV settings but even broader settings.

THEOREM 6.6. *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 a set of egds. For every existential query q over the target schema $\mathbf{T}$, there is a polynomial-time algorithm for computing, given a source instance I, the set* u-certain(q, I) *of the certain answers of q on the universal solutions for I.*

Proof: Let q be an existential query, and let J_0 be the core of the universal solutions. We see from Proposition 6.4 that u-certain$(q, I) = q(J_0)_{\downarrow}$. By Theorem 5.2 or Theorem 5.15, there is a polynomial-time algorithm for computing J_0, and hence for computing $q(J_0)_{\downarrow}$. ∎

Theorems 6.5 and 6.6 show a computational advantage for certain answers on universal solutions over simply certain answers. Note that the core is used in the proof of Theorem 6.6 but does not appear in the statement of the theorem and does not enter into the definitions of the concepts used in the theorem. It is not at all clear how one would prove this theorem directly, without making use of our results about the core.

We close this section by pointing out that Proposition 6.4 is very dependent on the assumption that q is an existential query. A *universal query* is taken to be the negation of an existential query. It is a query of the form $\forall \mathbf{x} \phi(\mathbf{x})$, where $\phi(\mathbf{x})$ is a quantifier-free formula, with a safety condition that is inherited from existential queries. Note that each egd and full tgd is a universal query (and in particular, satisfies the safety condition). For example, the egd $\forall \mathbf{x}(A_1 \wedge A_2 \rightarrow (x_1 = x_2))$ satisfies the safety condition, since its negation is $\exists \mathbf{x}(A_1 \wedge A_2 \wedge (x_1 \neq x_2))$, which satisfies the safety condition for existential queries since every variable in $\mathbf{x}$ appears in one of the atomic formulas A_1 or A_2.

We now give a data exchange setting and a universal query q such that u-certain(q, I) cannot be obtained by evaluating q on the core of the universal solutions for I.

EXAMPLE 6.7. Referring to our running example, consider again the universal solutions J_m, for $m \geq 0$, from Example 6.1. Among those universal solutions, the instance J_0 is the core of the universal solutions for I.

Let q be the following Boolean universal query (a functional dependency):

$$\forall e \forall d_1 \forall d_2 (\texttt{EmpDept}(e, d_1) \wedge \texttt{EmpDept}(e, d_2) \rightarrow (d_1 = d_2)).$$

It is easy to see that $q(J_0) = \underline{\mathsf{true}}$ and $q(J_m) = \underline{\mathsf{false}}$, for all $m \geq 1$. Consequently,

$$\textit{certain}(q, I) = \underline{\mathsf{false}} = \text{u-certain}(q, I) \neq q(J_0).$$

7 Concluding Remarks

In a previous paper [Fagin et al. 2003], 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 identified a particular universal solution, namely, the core of an arbitrary universal solution, and argued 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 existential queries. 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, we proposed an alternative semantics where the set of "certain answers" are redefined to be those that occur in every universal solution. Under this alternative semantics, the core gives the exact answer for existential queries.

We considered the question of the complexity of computing the core. To this effect, we showed that the complexity of deciding if a graph H is the core of a graph G is DP-complete. Thus, 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 gave natural conditions where there are polynomial-time algorithms for computing the core of universal solutions. Specifically, we showed that the core of the universal solutions is polynomial-time computable in data exchange settings in which Σ_{st} is a set of source-to-target tgds and Σ_t is a set of egds.

These results raise a number of questions. First, 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, the exponent may be somewhat large. Is there a more efficient algorithm for computing the core in this case and, if so, what is the most efficient such algorithm? There is also the question of extending the polynomial-time result to broader classes of target dependencies. To this effect, Gottlob [Gottlob 2004] recently showed that computing the core may be NP-hard in the case in which Σ_t consists of a single full tgd, provided a NULL "built-in" target predicate is available to tell labelled nulls from constants in target instances; note that, since NULL is a "built-in" predicate, it need not be preserved under homomorphisms. Since our formalization of data exchange does not allow for such a NULL predicate, it remains an open problem to determine the complexity of computing the core in data exchange settings in which the target constraints are egds and 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 [Deutsch et al. 1999].

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 [Fagin et al. 2003] can be extended to the more general case of XML/nested data exchange.

Acknowledgments. Many thanks to Marcelo Arenas, Georg Gottlob, Renée J. Miller, Arnon Rosenthal, Wang-Chiew Tan, Val Tannen and Moshe Y. Vardi for helpful suggestions, comments, and pointers to the literature.

REFERENCES

ABITEBOUL, S. AND DUSCHKA, O. M. 1998. Complexity of Answering Queries Using Materialized Views. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. 254–263.

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases.* Addison-Wesley.

BEERI, C. AND VARDI, M. Y. 1984. A Proof Procedure for Data Dependencies. *Journal of the ACM 31,* 4, 718–741.

CHANDRA, A. K. AND MERLIN, P. M. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*. 77–90.

COSMADAKIS, S. 1983. The Complexity of Evaluating Relational Queries. *Information and Control 58*, 101–112.

COSMADAKIS, S. S. AND KANELLAKIS, P. C. 1986. Functional and Inclusion Dependencies: A Graph Theoretic Approach. In *Advances in Computing Research*. Vol. 3. JAI Press, 163–184.

DEUTSCH, A., POPA, L., AND TANNEN, V. 1999. Physical Data Independence, Constraints and Optimization with Universal Plans. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. 459–470.

DEUTSCH, A. AND TANNEN, V. 2003. Reformulation of XML Queries and Constraints. In *Proceedings of the International Conference on Database Theory (ICDT)*. 225–241.

FAGIN, R. 1982. Horn Clauses and Database Dependencies. *Journal of the ACM 29,* 4 (Oct.), 952–985.

FAGIN, R., KOLAITIS, P. G., MILLER, R. J., AND POPA, L. 2003. Data Exchange: Semantics and Query Answering. In *Proceedings of the International Conference on Database Theory (ICDT)*. 207–224.

FRIEDMAN, M., LEVY, A. Y., AND MILLSTEIN, T. D. 1999. Navigational Plans For Data Integration. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 67–73.

GOTTLOB, G. 2004. Cores for data exchange: Hard cases and practical solutions. Manuscript.

GOTTLOB, G. AND FERMÜLLER, C. 1993. Removing redundancy from a clause. *Artificial Intelligence 61,* 2, 263–289.

HALEVY, A. 2001. Answering Queries Using Views: A Survey. *VLDB Journal*, 270–294.

HELL, P. AND NEŠETŘIL, J. 1992. The Core of a Graph. *Discrete Mathematics 109*, 117–126.

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

LENZERINI, M. 2002. Data Integration: A Theoretical Perspective. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. 233–246.

MAIER, D., MENDELZON, A. O., AND SAGIV, Y. 1979. Testing Implications of Data Dependencies. *ACM Transactions on Database Systems (TODS) 4,* 4 (Dec.), 455–469.

MILLER, R. J., HAAS, L. M., AND HERNÁNDEZ, M. 2000. Schema Mapping as Query Discovery. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. 77–88.

PAPADIMITRIOU, C. AND YANNAKAKIS, M. 1982. The Complexity of Facets and Some Facets of Complexity. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*. 229–234.

PAPADIMITRIOU, C. H. 1994. *Computational Complexity*. Addison-Wesley.

POPA, L., VELEGRAKIS, Y., MILLER, R. J., HERNANDEZ, M. A., AND FAGIN, R. 2002. Translating Web Data. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. 598–609.

SHU, N. C., HOUSEL, B. C., TAYLOR, R. W., GHOSH, S. P., AND LUM, V. Y. 1977. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. *ACM Transactions on Database Systems (TODS) 2,* 2, 134–174.

VAN DER MEYDEN, R. 1998. Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems*. Kluwer, 307–356.