Peer Data Exchange

Ariel Fuxman
U. of Toronto
afuxman@cs.toronto.edu

Phokion G. Kolaitis*
IBM Almaden
kolaitis@almaden.ibm.com

Renée J. Miller
U. of Toronto
miller@cs.toronto.edu

Wang-Chiew Tan†
UC Santa Cruz
wctan@cs.ucsc.edu

ABSTRACT

In this paper, we introduce and study a framework, called *peer data exchange*, for sharing and exchanging data between peers. This framework is a special case of a full-fledged peer data management system and a generalization of data exchange between a source schema and a target schema. The motivation behind peer data exchange is to model authority relationships between peers, where a source peer may contribute data to a target peer, specified using source-to-target constraints, and a target peer may use target-to-source constraints to restrict the data it is willing to receive, but cannot modify the data of the source peer.

A fundamental algorithmic problem in this framework is that of deciding the existence of a solution: given a source instance and a target instance for a fixed peer data exchange setting, can the target instance be augmented in such a way that the source instance and the augmented target instance satisfy all constraints of the setting? We investigate the computational complexity of the problem for peer data exchange settings in which the constraints are given by tuple generating dependencies. We show that this problem is always in NP, and that it can be NP-complete even for "acyclic" peer data exchange settings. We also show that the data complexity of the certain answers of target conjunctive queries is in coNP, and that it can be coNP-complete even for "acyclic" peer data exchange settings.

After this, we explore the boundary between tractability and intractability for the problem of deciding the existence of a solution. To this effect, we identify broad syntactic conditions on the constraints between the peers under which testing for solutions is solvable in polynomial time. These syntactic conditions include the important special case of peer data exchange in which the source-to-target constraints are arbitrary tuple generating dependencies, but the target-to-source constraints are local-as-view dependencies. Finally, we show that the syntactic conditions we identified are tight, in the sense that minimal relaxations of them lead to intractability.

*On leave from UC Santa Cruz.

†Supported in part by an NSF CAREER award IIS-0347065 and an NSF grant IIS-0430994.

PODS 2005 June 13-15, 2005, Baltimore, Maryland.

1. Introduction

Several different frameworks for sharing data between independent stores have been formulated and investigated in depth. *Data exchange* is one of the conceptually simpler, yet technically challenging, such frameworks [8]. In a data exchange setting, data from a source schema are transformed to data over a target schema according to specifications given by source-to-target constraints. This framework models a situation in which the target passively receives data from the source, as long as the source-to-target constraints are satisfied. Data exchange is closely related to data integration [15]. In particular, data exchange systems can be used as building blocks in data integration systems, where data from a set of independent sources having no interaction with each other are transformed to data in a global mediated schema. *Peer data management systems* (PDMS) constitute a much more powerful and complex framework than data exchange, as they model a situation in which a number of peers interact with each other and cooperate in sharing and exchanging data [14, 20, 19]. In a peer data management system, there is no distinction between source and target, since a peer may simultaneously act as a distributor of data (thus, a source peer) and a recipient of data (thus, a target peer). In such a system, the relationship between peers is specified using constraints that can be in either direction (from one peer to another, and vice-versa), instead of constraints in a single direction, as was the case in data exchange. Furthermore, each peer can be a stand-alone database system or a separate data integration system in which the schema of the peer is a mediated global schema over a set of local sources accessible only by that peer.

The Peer Data Exchange Framework In this paper, we introduce and study a framework, called *peer data exchange*, which is a generalization of data exchange and a special case of a full-fledged peer data management system. This framework models a situation in which there is interaction between two peers that have different roles and capabilities: one of them, called the *source* peer, is an "authoritative" or "trusted" peer that can contribute new data, while the other peer, called the *target* peer, imposes restrictions on the data that it is willing to accept, but has no permission or capability to modify the data of the source peer. In a peer data exchange setting, the relationship between the two peers is specified by constraints that go in either direction, that is, some are source-to-target constraints and others are target-to-source constraints. As in data exchange, the source-to-target constraints specify what data a source peer is willing to exchange. Unlike data exchange, however, the target is no longer a passive recipient of source data that

obey the source-to-target constraints. Instead, the target peer uses target-to-source constraints to impose restrictions on the data that it is willing to receive; moreover, the target may have its own data. Suppose that we are given a source instance and a target instance that may or may not satisfy the constraints of the setting; if the constraints are not satisfied, the goal then is to augment the target data in such a way that the *given* source instance and the *augmented* target instance satisfy all constraints between the two peers, as well as other existing target constraints. As an illustration, the source peer may be an authoritative genomic database, such as Swiss-Prot [17], while the target peer may be a genomic database maintained at a university under a different schema and populated with various data. At regular intervals of time, the university database is willing to receive new data from Swiss-Prot but cannot export any data back to Swiss-Prot. The target may restrict the data it is willing to receive to only Swiss-Prot data that it views as relevant. Hence, the data received have to satisfy constraints that go in either direction.

Algorithmic Problems The first fundamental algorithmic problem in peer data exchange is that of deciding the *existence of a solution*. More formally, a peer data exchange setting consists of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, a set of source-to-target constraints Σ_{st}, a set of target-to-source constraints Σ_{ts}, and a set Σ_t of target constraints. Each such setting, gives rise to the following decision problem: given a source instance and a target instance, can the target instance be augmented in such a way that the given source instance and the augmented target instance satisfy all constraints of the peer exchange setting? The second fundamental algorithmic problem in peer data exchange is that of obtaining *the certain answers* of queries posed over the target schema. The concept of the certain answers has become the standard semantics of query-answering in data integration [1, 15], data exchange [8], and peer data management [14]; this concept is also perfectly meaningful in peer data exchange.

In the sequel, we investigate these algorithmic problems for peer data exchange settings in which the constraints between the peers are given by a finite set of tuple-generating dependencies (tgds) [3]. We also allow for target constraints in the form of target tgds or target equality-generating dependencies (target egds). By definition, a tgd from one relational schema to another is a first-order formula of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x},\mathbf{y}))$, where $\varphi(\mathbf{x})$ is a conjunction of atomic formulas over the first schema and $\psi(\mathbf{x},\mathbf{y})$ is a conjunction of atomic formulas over the second. An equality-generating dependency on a relational schema is a formula of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow z_1 = z_2)$, where $\varphi(\mathbf{x})$ is a conjunction of atomic formulas over the schema and z_1, z_2 are among the variables in $\mathbf{x}$. Tuple-generating dependencies have been used for specifying data exchange between relational schemas [8, 7]; moreover, they are the core of the mapping specification language of the Clio schema-mapping and data exchange system [18]. Tuple-generating dependencies generalize both the local-as-view (LAV) and the global-as-view (GAV) constraints in data integration [15], since the former are tgds in which $\varphi(\mathbf{x})$ is a single atomic formula, and the latter are tgds in which $\psi(\mathbf{x},\mathbf{y})$ is a single atomic formula. In their full generality, tuple-generating dependencies are GLAV (*global-and-local-as-view*) constraints.

Summary of Results Consider a fixed peer data exchange setting in which Σ_{st} is a finite set of source-to-target tgds, Σ_{ts} is a finite set of target-to-source tgds, and $\Sigma_t = \emptyset$ (no target constraints). Our first main result asserts that testing for the existence of solutions is in NP, and that the data complexity of the certain answers of unions of conjunctive queries is in coNP. These complexity bounds turn out to be tight, because we exhibit peer data exchange settings as above for which testing for the existence of solutions is NP-complete, while the data complexity of the certain answers of conjunctive queries is coNP-complete; actually, the lower bounds hold even for peer data exchange settings in which the "dependency" graph between the relations of the peers is acyclic. We also show that the same upper bounds hold even if the setting allows for a set Σ_t of target constraints that is the union of a finite set of target egds and a finite *weakly acyclic* set of target tgds.

The complexity of testing for the existence of solutions and computing the certain answers in peer data exchange settings should be compared and contrasted with the complexity of the same problems for data exchange, which can be viewed as the special case of peer data exchange in which $\Sigma_{ts} = \emptyset$ (no target-to-source tgds) and also $J = \emptyset$ (the target contains no data before the exchange). Indeed, as shown in [8], there are polynomial-time algorithms to test for the existence of solutions and to compute the certain answers of unions of conjunctive queries in every data exchange setting in which Σ_{st} is a finite set of source-to-target tgds and Σ_t is the union of a finite set of target egds and a finite weakly acyclic set of target tgds. Moreover, if $\Sigma_t = \emptyset$ (no target constraints), then testing for the existence of solutions is trivial for data exchange, since solutions always exist. There is also a sharp contrast with full-fledged peer data management systems, where, as shown in [14], computing the certain answers of conjunctive queries can be an undecidable problem. Thus, from a computational point of view, peer data exchange is more challenging than ordinary data exchange, but less intractable than full peer data management.

After this, we explore the boundary between tractability and intractability in peer data exchange settings.To this effect, we identify a class of peer data exchange settings, denoted by $\mathcal{C}_{tract}$, for which the existence of solutions can be tested in polynomial time. The class $\mathcal{C}_{tract}$ is defined by imposing syntactic conditions on the constraints between the peers; these conditions are extracted through a careful examination of the impact of existentially quantified variables and of their relationship to other variables occurring in the constraints. Although the syntactic conditions used to define $\mathcal{C}_{tract}$ are rather technical, $\mathcal{C}_{tract}$ itself is a broad class that contains several important special cases of peer data exchange, including the following two: the case in which the source-to-target tgds are *full* tgds, and the case in which the target-to-source tgds are local-as-view (LAV) constraints. Finally, we show that the syntactic conditions we identified are tight, in the sense that minimal relaxations of the conditions lead to intractability; thus $\mathcal{C}_{tract}$ turns out to be a maximal class of tractable peer data exchange settings.

Related Work There is an extensive literature on data integration using sound, complete and exact views [1, 13, 15]. Several different frameworks and systems for sharing data in networks of independent sources have also been formulated and studied [4, 16, 6, 10, 11]. Calvanese et al. [6] and Franconi et al. [10, 11] propose a semantics based on an epistemic interpretation of the constraints between peers. This is in contrast to the first-order interpretation used in our work and in PDMS. Bertossi and Bravo [5] also use first-order interpretations, but propose a semantics drawn from the area of consistent query answering that is based on repairs [2]. This approach has the advantage that data can be shared between peers, even when there is no consistent solution satisfying all constraints. However, the complexity of the problem of obtaining certain answers is higher than in peer data exchange (Π_2^p-complete vs. coNP-

complete), and no tractability results have been given for this semantics.

2. Peer Data Exchange Settings

This section contains the precise definitions of a peer data exchange setting and the associated algorithmic problems, as well as a brief discussion of the relationship of peer data exchange settings with data exchange settings and peer data management systems.

Preliminaries

A *schema* is a finite collection $\mathbf{R} = (R_1, \ldots, R_k)$ of relation symbols, each of a fixed arity. An *instance* I over $\mathbf{R}$ is a sequence $(R_1^I, \ldots, R_k^I)$ such that each R_i^I is a finite relation of the same arity as R_i. We shall often use R_i to denote both the relation symbol and the relation R_i^I that interprets it. Given a tuple $\mathbf{t}$, we denote by $R(\mathbf{t})$ the association between $\mathbf{t}$ and the relation R where it occurs. Let $\mathbf{S} = (S_1, \ldots, S_n)$ and $\mathbf{T} = (T_1, \ldots, T_m)$ be two disjoint schemas. We refer to $\mathbf{S}$ as the *source* schema and to $\mathbf{T}$ as the *target* schema. We write $(\mathbf{S}, \mathbf{T})$ to denote the schema $(S_1, \ldots, S_n, T_1, \ldots, T_m)$. Instances over $\mathbf{S}$ will be called *source* instances, while instances over $\mathbf{T}$ will be called *target* instances. If I is an instance over $\mathbf{S}$ and J is an instance over $\mathbf{T}$, then we write (I, J) to denote the instance K over $(\mathbf{S}, \mathbf{T})$ such that $S_i^K = S_i^I$ and $T_j^K = T_j^J$, for $1 \leq i \leq n$ and $1 \leq j \leq m$.

A *source-to-target tuple-generating dependency* (tgd) is a formula of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$, where $\varphi(\mathbf{x})$ is a conjunction of atomic formulas over the source schema $\mathbf{S}$, and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over the target schema $\mathbf{T}$. Similarly, a *target-to-source tgd* is a formula of the form $\forall \mathbf{x}(\alpha(\mathbf{x}) \rightarrow \exists \mathbf{y} \beta(\mathbf{x}, \mathbf{y}))$, where $\alpha(\mathbf{x})$ is a conjunction of atomic formulas over the target schema $\mathbf{T}$, and $\beta(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over the target schema $\mathbf{S}$. For example, if $\mathbf{S}$ contains a binary relation E, and $\mathbf{T}$ contains a binary relation H, then the source-to-target tgd $\forall x \forall y \forall z (E(x, z) \wedge E(z, y) \rightarrow H(x, y))$ transforms pairs of nodes connected via an E-path of length 2 to H-edges. Similarly, $\forall x \forall y (H(x, y) \rightarrow \exists z (E(x, z) \wedge E(z, y)))$ is a target-to-source tgd that transforms H-edges to pairs of nodes connected via an E-path of length 2.

A *target tgd* is a formula of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \chi(\mathbf{x}, \mathbf{y}))$, where both $\varphi(\mathbf{x})$ and $\chi(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over the target schema $\mathbf{T}$. A *target equality-generating dependency* (egd) is a formula of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow z_1 = z_2)$, where $\varphi(\mathbf{x})$ is a conjunction of atomic formulas over $\mathbf{T}$ and z_1, z_2 are among the variables in $\mathbf{x}$. Clearly, functional dependencies on $\mathbf{T}$ are special cases of target egds. In what follows, we will often drop the universal quantifiers in front of a dependency, and implicitly assume such quantification. However, we will write down all existential quantifiers.

Peer Data Exchange Settings and Solutions

DEFINITION 1. A *peer data exchange (PDE) setting* is a quintuple $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \Sigma_t)$ such that:

- $\mathbf{S}$ is a source schema and $\mathbf{T}$ is a target schema;
- Σ_{st} is a finite set of source-to-target tgds;
- Σ_{ts} is a finite set of target-to-source tgds;
- Σ_t is a finite set of target tgds and target egds.

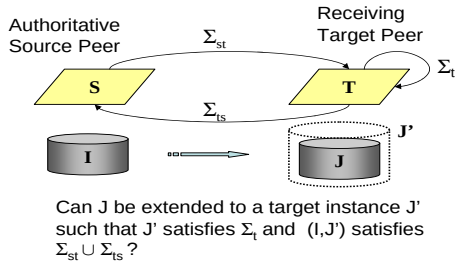

Figure 1: Illustration of Peer Data Exchange

Given a source instance I and a target instance J of $\mathcal{P}$, it may be the case that (I, J) violates the constraints of $\mathcal{P}$. Thus, we will be interested in finding instances, which we call *solutions*, that satisfy all constraints of $\mathcal{P}$. In peer data exchange the target peer is assumed to be willing to accept data coming from an authoritative, trusted source. Therefore, we will consider solutions where the instance of the target peer may be augmented with data coming from the source. However, the target peer does not have the authority or ability to interfere with the source's data, which therefore remain unchanged.

DEFINITION 2. Let $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \Sigma_t)$ be a PDE setting, I a source instance, and J a target instance. We say that a target instance J' is a *solution for* (I, J) *in* $\mathcal{P}$ if

- $J \subseteq J'$;
- $(I, J') \models \Sigma_{st} \cup \Sigma_{ts}$;
- $J' \models \Sigma_t$.

This definition generalizes the notion of solution in data exchange settings [8] in two ways. The first and more significant one is the presence of the target-to-source dependencies Σ_{ts} ; the second is that the input has a target instance J, in addition to the source instance I. Thus, data exchange settings are a special case of PDE settings where both Σ_{ts} and J are empty.

As noted earlier, tuple-generating dependencies are GLAV constraints that generalizes both LAV and GAV constraints in data integration systems. Our PDE framework, with target-to-source dependencies, is able to capture GLAV with exact views in data integration systems [15]. The following source-to-target dependency $\phi(\mathbf{x}) \rightarrow \exists y\, \psi(\mathbf{x}, \mathbf{y})$ and target-to-source dependency $\psi(\mathbf{x}, \mathbf{y}) \rightarrow \phi(\mathbf{x})$, where ϕ and ψ can be interpreted as queries over the source and target respectively, assert that the query over the target contains exactly those tuples from the query over the source. It is easy to see that in the case where ϕ is a single source relation, this expresses LAV with exact views in data integration.

Although the definition of PDE setting involves two peers, it can be easily extended to a family of source peers exchanging data with the same target peer. Assume that $\mathbf{S}_1, \ldots, \mathbf{S}_n, \mathbf{T}$ are pairwise disjoint schemas. A *multi-PDE setting* is a family $\mathcal{P}_1 = (\mathbf{S}_1, \mathbf{T}, \Sigma_{s_1 t}, \Sigma_{t s_1}, \Sigma_{t_1})$, ..., $\mathcal{P}_n = (\mathbf{S}_n, \mathbf{T}, \Sigma_{s_n t}, \Sigma_{t s_n}, \Sigma_{t_n})$ of PDE settings. Given instances $I_1, \ldots, I_n$ of the source peers, and an instance J of the target peer, a *solution* J' *for* $((I_1, \ldots, I_n), J)$ *in* $\mathcal{P}_1, \ldots, \mathcal{P}_n$ is a target instance J' containing J such that J' is a solution for (I_m, J) in $\mathcal{P}_m$, for every $m \leq n$. Note that, in defining multi-PDE settings, we could have allowed constraints on the sources $\mathbf{S}_1, \ldots, \mathbf{S}_n$, as well as constraints between these sources.

This, however, would have no impact on which target instances are solutions, as the source instances have to remain unchanged.

It is clear that J' is a solution for $((I_1, \ldots, I_n), J)$ in $\mathcal{P}_1, \ldots, \mathcal{P}_n$ if and only if J' is a solution for $(I_1 \cup \cdots \cup I_n, J)$ in the PDE setting $(\mathbf{S}_1 \cup \cdots \cup \mathbf{S}_n, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \Sigma_t)$, where $\Sigma_{st} = \cup_{m=1}^{n} \Sigma_{s_m t}$, $\Sigma_{ts} = \cup_{m=1}^{n} \Sigma_{t s_m}$, and $\Sigma_t = \cup_{m=1}^{n} \Sigma_{t_m}$. Thus, every multi-PDE setting can be simulated by a single PDE that has the same space of solutions as the original multi-PDE.

Algorithmic Problems in PDE Settings

Given a source instance I and a target instance J of a PDE setting $\mathcal{P}$, a solution for (I, J) may or may not exist; furthermore, if a solution exists, it need not be unique up to isomorphism.

EXAMPLE 1. Let $\mathcal{P}$ be a PDE setting in which the source schema consists of a binary relation symbol E, the target schema consists of a binary relation symbol H, and the constraints are as follows:

$$\begin{array}{ll} \Sigma_{st}: & E(x,z) \wedge E(z,y) \rightarrow H(x,y) \\ \Sigma_{ts}: & H(x,y) \rightarrow E(x,y) \\ \Sigma_t: & \emptyset \quad \text{(no target constraints)} \end{array}$$

If $I = \{E(a,b), E(b,c)\}$ and $J = \emptyset$, then no solution for (I, J) exists. If $I = \{E(a,a)\}$ and $J = \emptyset$, then $J' = \{H(a,a)\}$ is the only solution for (I, J). If $I = \{E(a,b), E(b,c), E(a,c)\}$ and $J = \emptyset$, then both $\{H(a,c)\}$ and $\{H(a,b), H(b,c), H(a,c)\}$ are solutions for (I, J). □

This example illustrates a striking difference between data exchange settings and peer data exchange settings. Specifically, if a data exchange setting has no target constraints ($\Sigma_t = \emptyset$), then, for every source instance I, a solution always exists. As seen above, however, this need not be true for peer data exchange settings with $\Sigma_t = \emptyset$ and $J = \emptyset$. We will study in depth the problem of deciding the existence of a solution in a peer data exchange settings, and we will unveil deeper differences between data exchange and peer data exchange.

DEFINITION 3. Assume that $\mathcal{P}$ is a PDE setting. The *existence-of-solutions problem for* $\mathcal{P}$, denoted by $\mathtt{SOL}(\mathcal{P})$, is the following decision problem: given a source instance I and a target instance J, is there a solution J' for (I, J) in $\mathcal{P}$?

The other basic algorithmic problem that we will study is that of obtaining the *certain answers* of target queries in PDE settings. The definition of certain answers we use is an adaptation of the standard concept used in incomplete databases [12, 21] and information integration [1, 15]; in our context, this means that the set of "possible" worlds is the set of all solutions for a given source instance and a given target instance in a PDE setting.

DEFINITION 4. Let $\mathcal{P}$ be a PDE setting and q a query over the target schema of $\mathcal{P}$. Let also I be a source instance and J a target instance. We say that a tuple $\mathbf{t}$ is a *certain answer of* q *on* (I, J), denoted $\mathbf{t} \in \textbf{certain}(q, (I, J))$, if $J' \models q[\mathbf{t}]$, for every solution J' for (I, J) in $\mathcal{P}$. We write $\textbf{certain}(q, (I, J))$ to denote the set of all certain answers of q on (I, J). If q is a Boolean query, then $\textbf{certain}(q, (I, J)) = \mathtt{true}$ if $J' \models q$, for every solution J' for (I, J) in $\mathcal{P}$; otherwise, $\textbf{certain}(q, (I, J)) = \mathtt{false}$. Note that if q is a Boolean query, then computing the certain answers of q in the PDE setting $\mathcal{P}$ is a decision problem.

Consider the PDE setting in Example 1. If q is the Boolean query $\exists x \exists y \exists z (H(x,y) \wedge H(y,z))$, then $\textbf{certain}(q, (\{E(a,a)\}, \emptyset)) = \mathtt{true}$, while $\textbf{certain}(q, (\{E(a,b), E(b,c), E(a,c)\}, \emptyset)) = \mathtt{false}$.

Relationship to PDMS

Peer data management systems (PDMS), formalized and studied by Halevy et al. [14], constitute a decentralized, extensible architecture in which peers interact with each other in sharing and exchanging data. As mentioned in the Introduction, every PDE setting is a special case of a PDMS. In this section, we describe the relationship between peer data exchange settings and peer data management systems in precise terms.

According to [14], a PDMS $\mathcal{N}$ with peers $P_1, \ldots, P_n$ has the following characteristics.

• Each peer P_i has its own schema which is disjoint from those of the other peers, but visible to all other peers.

• The schema of each peer can be a mediated global schema over a set of local sources that are accessible only by that peer (thus each peer can be a data integration system). The relationship between the peer and its local sources is specified using *storage descriptions* that are *containment descriptions* $R \subseteq Q$ or *equality descriptions* $R = Q$, where R is one of the relations in the schema of the peer and Q is a query over the local sources of the peer.

• The relationship between peers is specified using three types of *peer mappings*: *inclusion mappings*, *equality mappings*, and *definitional mappings*, where

1. Each inclusion mapping is a containment $Q_1(\mathcal{A}_1) \subseteq Q_2(\mathcal{A}_2)$ between conjunctive queries $Q_1(\mathcal{A}_1)$ and $Q_2(\mathcal{A}_2)$, where $\mathcal{A}_1$ and $\mathcal{A}_2$ are subsets of the set of all relations in the schemas of the peers.
2. Each equality mapping is an equality $Q_1(\mathcal{A}_1) = Q_2(\mathcal{A}_2)$ between conjunctive queries $Q_1(\mathcal{A}_1)$ and $Q_2(\mathcal{A}_2)$ as above.
3. Each definitional mapping is a Datalog program with rules having single relations from the schemas of the peers in both the head and the body of each rule.

In the terminology of [14], a *data instance* D of a PDMS $\mathcal{N}$ is an assignment of values to both the local sources of each peer and to the relations of the schema of each peer. A data instance G is *consistent* with $\mathcal{N}$ and D if G and D satisfy all the specifications given by the storage descriptions and the peer mappings of $\mathcal{N}$ (see [14] for the precise definition). This concept captures what it means for a data instance G to be a *solution* for a given data instance D in the PDMS $\mathcal{N}$.

We now have all the necessary background to spell out the relationship between peer data exchange settings and peer data management systems. Indeed, let $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \Sigma_t)$ be a PDE setting. We claim that there is a PDMS $\mathcal{N}(\mathcal{P})$ with two peers $\mathbf{S}$ and $\mathbf{T}$ such that the solutions for a given instance in $\mathcal{P}$ essentially coincide with the consistent data instances for a corresponding data instance in $\mathcal{N}(\mathcal{P})$. The specification of the PDMS $\mathcal{N}(\mathcal{P})$ is as follows:

• The peer mappings of $\mathcal{N}(\mathcal{P})$ are given by the dependencies in $\Sigma_{st} \cup \Sigma_{ts} \cup \Sigma_t$. In particular, $\mathcal{N}(\mathcal{P})$ has no definitional mappings.

• For every relation symbol S_i in the schema of $\mathbf{S}$, there is a local relation symbol S_i^* of the same arity as S_i, and an equality storage description $S_i^* = S_i$.

• For every relation symbol T_j in the schema of $\mathbf{T}$, there is a local relation symbol T_j^* of the same arity as T_j, and a containment storage description $T_j^* \subseteq T_j$.

Note that the schemas of the local sources of $\mathbf{S}$ and $\mathbf{T}$ in $\mathcal{N}(\mathcal{P})$ are replicas of the schemas of $\mathbf{S}$ and $\mathbf{T}$. Intuitively, the equality storage descriptions for $\mathbf{S}$ capture the fact that in peer data exchange the data of the source peer remain unchanged, whereas the containment storage descriptions for $\mathbf{T}$ capture the fact that in peer data exchange the data of the target peer may be augmented with new data. Let I be a source instance and let J be a target instance of $\mathcal{P}$. It is now easy to verify that K is a solution for (I, J) in $\mathcal{P}$ if and only if $(I^*, I), (J^*, K)$ is a consistent data instance for the data instance (I^*, J^*) of $\mathcal{N}(\mathcal{P})$, where I^* and J^* are copies of I and J over the local sources of $\mathbf{S}$ and $\mathbf{T}$.

In conclusion, every PDE setting can be viewed as a PDMS with equality storage descriptions $S_i^* = S_i$ for the source peer, containment storage descriptions $T_j^* \subseteq T_j$ for the target peer, and peer mappings given by the constraints of the PDE.

There are peer data management systems for which testing for the existence of solutions and computing the certain answers of conjunctive queries are undecidable problem as well [14]. We will show that the state of affairs is quite different for peer data exchange settings.

3. Complexity

Let $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \Sigma_t)$ be a fixed peer data exchange setting. In this section, we show that the existence-of-solutions problem for $\mathcal{P}$ is in NP, where Σ_{st} and Σ_{ts} are arbitrary finite sets of source-to-target tgds and target-to-source tgds, and Σ_t is assumed to be the union of a finite set of target egds with a *weakly acyclic* finite set of target tgds. For such settings, the data complexity of the certain answers of *monotone* queries (in particular, unions of conjunctive queries) is in coNP. We also show that there are PDE settings with $\Sigma_t = \emptyset$ for which the existence-of-solutions problem is NP-complete, and the data complexity of the certain answers of conjunctive queries is coNP-complete.

These results about peer data exchange settings contrast sharply both with results about peer data management systems and with results about data exchange settings. As mentioned earlier, there are PDMS for which these problems are undecidable [14]. For data exchange settings in which Σ_{st} is an arbitrary finite set of source-to-target tgds and Σ_t is the union of a finite set of target egds with a weakly acyclic finite set of target tgds (recall that in data exchange settings there are no target-to-source tgds), these problems are solvable in polynomial time [8]. In fact, if $\Sigma_t = \emptyset$, then the existence-of-solutions problem is trivial, as solutions always exists.

3.1 Upper Bound

The concept of a *weakly acyclic* set of target tgds was introduced in [8] and used to show that the *chase procedure* terminates in polynomial time on such sets of tgds. Intuitively, weak acyclicity is a syntactic condition placed on sets of tgds to ensure that a chase step does not use labeled nulls from an attribute to create new labeled nulls in the same attribute. This ensures that the chase sequence is finite.

DEFINITION 5. [8] **(Weakly acyclic set of tgds)** Let Σ be a set of tgds over a fixed schema. Construct a directed graph, called the *dependency graph*, as follows: (1) there is a node for every pair (R, A) with R a relation symbol of the schema and A an attribute of R; call such a pair (R, A) a *position*; (2) add edges as follows: for every tgd $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})$ in Σ and for every x in $\mathbf{x}$ that **occurs** in ψ:

- For every occurrence of x in ϕ in position (R, A_i):
  1. for every occurrence of x in ψ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$ (if it does not already exist).
  2. in addition, for every existentially quantified variable y and for every occurrence of y in ψ in position (T, C_k), add a *special edge* $(R, A_i) \rightarrow (T, C_k)$ (if it does not already exists).

  Note that there may be two edges in the same direction between two nodes but exactly one of the two edges is special. Then Σ is *weakly acyclic* if the dependency graph has no cycle going through a special edge.

It should be noted that weakly acyclic sets of tgds include as a special case sets of full tgds, that is, tgds of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x}))$ in which no existentially quantified variables occur in the right-hand side. They also include acyclic sets of inclusion dependencies as a special case.

To obtain the complexity upper bounds, we extend the chase procedure in [8] to what we call a *solution-aware* chase procedure that chases an instance with tgds and with another given instance. This procedures chases an instance K with a set of tgds and at the same time uses values from a given instance K' (thought of as a "solution") that contains K and satisfies the tgds at hand. Instead of creating labeled nulls to witness the existential variables of a tgd during a chase step, a *solution-aware chase step* uses values from the given "solution" K' to witness the existential variables. These values are guaranteed to exist since K' contains K and satisfies the tgds. Note that values from K' are used only when a chase step is applied with a tgd that contains existential variables. The following is the definition of *solution-aware chase step* and *solution-aware chase sequence*.

DEFINITION 6. **(Solution-aware chase step)** Let K_1 be an instance.

(tgd) Let d be a tgd $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$. Let K be an instance that contains K_1 such that K satisfies d. Let h be a homomorphism from $\phi(\mathbf{x})$ to K_1 such that there is no extension of h to a homomorphism h' from $\phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y})$ to K_1. We say that *d can be applied to K_1 with homomorphism h and solution K*, or simply, *d can be applied to K_1 with homomorphism h* if K is understood from context.

Let K_2 be the union of K_1 with the set of facts obtained by taking the image of ψ under a homomorphism h' where h' is an extension of h such that each variable in $\mathbf{y}$ is assigned a value in K and the image of the atoms of ψ under h' are atoms in K. We say that *the result of applying d to K_1 with h and solution K is K_2*, and write $K_1 \xrightarrow{d,h,K} K_2$. We drop K and write $K_1 \xrightarrow{d,h} K_2$ if K is understood from context.

(egd) Let d be an egd $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow (x_1 = x_2))$. Let h be a homomorphism from $\phi(\mathbf{x})$ to K_1 such that $h(x_1) \neq h(x_2)$. We say that *d can be applied to K_1 with homomorphism h*. We distinguish two cases.

- If both $h(x_1)$ and $h(x_2)$ are in Const then we say that *the result of applying d to K_1 with h is "failure"*, and write $K_1 \xrightarrow{d,h} \perp$.
- Otherwise, let K_2 be K_1 where we identify $h(x_1)$ and $h(x_2)$ as follows: if one is a constant, then the labeled null is replaced everywhere by the constant; if both are labeled nulls, then one is replaced everywhere by the other. We say that *the result of applying d to K_1 with h is K_2*, and write $K_1 \xrightarrow{d,h} K_2$.

DEFINITION 7. **(Solution-aware chase)** Let Σ be a set of tgds and egds. Let K be an instance and K' be an instance that contains K and satisfies the set of tgds in Σ.

- A *solution-aware chase sequence of K with Σ and K'* is a sequence (finite or infinite) of solution-aware chase steps $K_i \xrightarrow{d_i,h_i} K_{i+1}$, with $i = 0, 1, ...$, with $K = K_0$ and d_i a dependency in Σ.
- A *finite solution-aware chase of K with Σ and K'* is a finite solution-aware chase sequence $K_i \xrightarrow{d_i,h_i} K_{i+1}$, $0 \leq i \leq m$, with the requirement that either (a) $K_m = \perp$ or (b) there is no dependency d_i of Σ and there is no homomorphism h_i such that d_i can be applied to K_m with h_i. We say that K_m is the result of the finite solution-aware chase. We refer to case (a) as the case of a *failing finite solution-aware chase* and we refer to case (b) as the case of a *successful finite solution-aware chase*.

LEMMA 1. *Let Σ be the union of a finite set of egds with a weakly acyclic finite set of tgds on some schema. Then there exists a polynomial $p(x)$ having the following property: if K and K' are instances such that K' contains K, and such that K' satisfies the tgds in Σ, then the length of every solution-aware chase sequence of K with Σ and K' is bounded by $p(|K|)$, where $|K|$ is the size of K.*

Using Lemma 1, we can show that whenever a solution for (I, J) exists in a PDE in which Σ_t is the union of a finite set of egds with a weakly acyclic finite set of tgds, then a "small" solution must exist, where "small" means that its size is polynomially bounded by the size of (I, J).

LEMMA 2. *Let $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \Sigma_t)$ be a PDE setting in which Σ_t is the union of a finite set of egds with a weakly acyclic finite set of tgds. Let I be a source instance and J be a target instance such that J satisfies Σ_t. If there exists a solution J' for (I, J), then there exists a solution J^* for (I, J) that is contained in J' and has size bounded by a polynomial in the size of (I, J).*

Using Lemmas 1 and 2, we can easily derive the following result.

THEOREM 1. *Let $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \Sigma_t)$ be a PDE setting in which Σ_t is the union of a finite set of egds with a weakly acyclic finite set of tgds. The existence-of-solutions problem $\mathsf{SOL}(\mathcal{P})$ for $\mathcal{P}$ is in NP.*

PROOF. From Lemma 2, if there is a solution for (I, J), then there is a solution (I, J^*) that is polynomial in the size of (I, J). Checking that $(I, J^*) \models \Sigma_{st}$, $(J^*, I) \models \Sigma_{ts}$ and $J^* \models \Sigma_t$ can be done in polynomial time in the size of (I, J) since the peer data exchange is fixed. □

By definition, a query q is *monotone* if it is preserved under the addition of tuples, that is, if $\mathbf{t} \in q(K)$ and $K \subseteq K'$, then $\mathbf{t} \in q(K')$. Clearly, unions of conjunctive queries are monotone queries.

THEOREM 2. *Let $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \Sigma_t)$ be a PDE setting in which Σ_t is the union of a finite set of egds with a weakly acyclic finite set of tgds. If q is a monotone query over $\mathbf{T}$, then computing the certain answers of q is in coNP.*

PROOF. Let $\mathbf{t}$ be a k-ary tuple from I and suppose $\mathbf{t} \notin \text{certain}(q, (I, J))$. It suffices to show that there is a solution J^* that is polynomial in the size of (I, J) and $t \notin q(J^*)$. Since $\mathbf{t} \notin \text{certain}(q, (I, J))$, there is a solution J' such that $\mathbf{t} \notin q(J')$. From Lemma 2, it follows that there is a solution J^* that is polynomial in the size of (I, J) and J^* is contained in J'. Since q is monotone and $\mathbf{t} \notin q(J')$, it follows that $\mathbf{t} \notin q(J^*)$. □

3.2 Lower Bound

We show next that there are PDE settings with no target constraints in which testing for the existence of solutions is NP-hard, and computing the certain answers of target conjunctive queries is coNP-hard. Although this result could be derived from [1, Theorem 5.1] and [13, Theorem 8], we give a self-contained proof using a particularly simple reduction from the CLIQUE problem whose features we will analyze later on.

THEOREM 3. *There exists a peer data exchange setting $\mathcal{P}$ with $\Sigma_t = \emptyset$ such that testing for the existence of solutions is a NP-complete problem. Moreover, there is a Boolean conjunctive query q such that the decision problem of computing the certain answers of q in $\mathcal{P}$ is coNP-complete.*

PROOF. *(Sketch)* From Theorem 1, we know that the problem is in NP. The NP-hardness is established via a reduction from the CLIQUE problem: given a graph G and a positive integer k, does G contain a k-clique? As usual, a *graph* is a structure $G = (V, E)$, where V is a set of nodes and $E \subseteq V^2$ is binary relation that is symmetric and irreflexive (no self-loops).

Let $\mathcal{P}$ be the following peer data exchange setting. The source schema $\mathbf{S}$ consists of three binary relations D, S and E, while the target schema $\mathbf{T}$ consists of a single 4-ary relation P. There are no target dependencies, that is, $\Sigma_t = \emptyset$. The constraints between $\mathbf{S}$ and $\mathbf{T}$ are as follows:

$$\begin{aligned} \Sigma_{st}: \quad & D(x,y) \rightarrow \exists z \exists w P(x,z,y,w) \\ \Sigma_{ts}: \quad & P(x,z,y,w) \rightarrow E(z,w) \\ & P(x,z,y,w) \wedge P(x,z',y',w') \rightarrow S(z,z') \end{aligned}$$

Given a graph $G = (V, E)$ and a positive integer k, we consider k distinct elements $a_1, \ldots, a_k$, and form the source instance $I(G, k) = (D, S, E)$, where $D = \{(a_i, a_j) \mid 1 \leq i \leq k, 1 \leq j \leq k, i \neq j\}$ is the inequality relation on $\{a_1, \ldots, a_k\}$ and $S = \{(v, v) \mid v \in V\}$ is the equality relation on the set V of nodes of G. The target instance J is defined to be empty. Intuitively, the tgd in Σ_{st} associates each pair of elements (x, y) in D with a pair of elements (z, w) through the relation P. The first tgd in Σ_{ts} asserts that (z, w) is an edge in E and the second tgd in Σ_{ts} asserts that an element in $a_1, \ldots, a_k$ cannot be associated with two distinct nodes in G.

It is now easy to verify that G has a k-clique if and only if there is a solution for $(I(G,k),\emptyset)$ in $\mathcal{P}$.

Let q be the Boolean query $\exists x P(x,x,x,x)$. We use the same reduction above for the coNP-hardness of the certain answers of q assuming that the k distinct elements are drawn from V, the node set of G. If V contains less than k nodes, one could extend V to k nodes. It is is easy to verify that G contains a k-clique if and only if certain$(q,(I(G,k),\emptyset)) = \texttt{false}$. □

In [14], it was shown that if in a PDMS all storage descriptions are containment descriptions and all peer mappings are inclusion mappings with an acyclic *dependency* graph, then the certain answers of conjunctive queries are computable in polynomial time. The *dependency* graph of a PDMS is the directed graph with nodes the relations of the peers, and edges between two relations P and R if there is an inclusion peer mapping $Q_1(\mathcal{A}_1) \subseteq Q_2(\mathcal{A}_2)$ such that P occurs in $Q_1(\mathcal{A}_1)$ and R occurs in $Q_2(\mathcal{A}_2)$. Note that the PDE setting used in the reduction of Theorem 3 has inclusion peer mappings with an acyclic dependency graph, yet the problem of computing certain answers is coNP-hard. The jump in complexity arises due to the fact that in PDE settings the source instance can never change, which means that the constraints placed on storage descriptions in the source are not containment descriptions, but equality descriptions.

4. A Large Tractable Class

In this section, we identify syntactic conditions on PDE settings with no target constraints that yield polynomial-time algorithms for deciding the existence of solutions. As seen in the proof of Theorem 3, even such strong topological conditions as the acyclicity of the dependency graph of source and target relations cannot guarantee tractability of these problems. Instead, we consider different conditions that are derived by taking a closer look at the existential quantifiers in the constraints of the PDE setting.

DEFINITION 8. Let $\mathcal{P} = (\mathbf{S},\mathbf{T},\Sigma_{st},\Sigma_{ts},\emptyset)$ be a PDE setting with no target constraints.
• We say that the i-th position of a relation symbol T of $\mathbf{T}$ is *marked* if Σ_{st} contains a source-to-target tgd

$$\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x},\mathbf{y})$$

such that $T(z_1,\ldots,z_i,\ldots,z_n)$ is one of the conjuncts of $\psi(\mathbf{x},\mathbf{y})$, and z_i is one of the existentially quantified variables $\mathbf{y}$.

• We say that a variable z is *marked in a target-to-source tgd*

$$\alpha(\mathbf{x}) \rightarrow \exists \mathbf{w}\beta(\mathbf{x},\mathbf{w})$$

of Σ_{ts} if one of the following two holds:

1. z appears at a marked position of a conjunct of $\alpha(\mathbf{x})$
2. z is one of the existentially quantified variables $\mathbf{w}$.

Note that the two conditions in the definition of a marked variable are mutually exclusive.

To illustrate the concepts of marked position and marked variable, let us consider a PDE setting having the following constraints:

$$\begin{array}{ll}\Sigma_{st}: & S(x_1,x_2) \rightarrow \exists y T(x_1,y)\\ \Sigma_{ts}: & T(x_1,x_2) \rightarrow \exists w S(w,x_2)\end{array}$$

In this setting, the only marked position is the second position of T, while the marked variables of the target-to-source dependency are x_2 and w.

Let us also consider the PDE setting in the proof of Theorem 3 used in the reduction from the CLIQUE problem:

$$\begin{array}{ll}\Sigma_{st}: & D(x,y) \rightarrow \exists z \exists w P(x,z,y,w)\\ \Sigma_{ts}: & P(x,z,y,w) \rightarrow E(z,w)\\ & P(x,z,y,w) \wedge P(x,z',y',w') \rightarrow S(z,z')\end{array}$$

In this setting, the marked positions are the second and the fourth position of P. The marked variables of the first tgd in Σ_{ts} are z and w, and the marked variables for the second tgd in Σ_{ts} are z, w, z', and w'.

We now introduce the class $\mathcal{C}_{tract}$, which is the focus of this section. Below, if $\alpha(\mathbf{x}) \rightarrow \exists \mathbf{w}\beta(\mathbf{x},\mathbf{w})$ is a tgd in Σ_{ts}, we will refer to $\alpha(\mathbf{x})$ as the *left-hand side* of the tgd, and to $\exists \mathbf{w}\beta(\mathbf{x},\mathbf{w})$ as the *right-hand side* of the tgd.

DEFINITION 9. Let $\mathcal{P} = (\mathbf{S},\mathbf{T},\Sigma_{st},\Sigma_{ts},\emptyset)$ be a PDE setting with no target constraints. We say that $\mathcal{P} \in \mathcal{C}_{tract}$ if

1. For every tgd D in Σ_{ts}, every marked variable of D appears at most once in the left-hand side of D

   and

2. One of the following two conditions holds:

   2.1 The left-hand side of every tgd in Σ_{ts} consists of exactly one literal;

   or

   2.2. For every tgd D in Σ_{ts} and for every pair of marked variables x and y of D that appear together in a conjunct of the right-hand side of D,

   either

   (a) x and y appear together in some conjunct of the left-hand side of D

   or

   (b) x and y do not appear at all in the left-hand side of D.

Admittedly, the definition of the class $\mathcal{C}_{tract}$ is quite technical. We arrived at it after carefully analyzing the causes of intractability in numerous concrete PDE settings, such as the one used in the reduction from the CLIQUE problem. To convey some feeling for $\mathcal{C}_{tract}$, we should point out that it is a rather broad class that contains several interesting families of PDE settings as subclasses.

Note that $\mathcal{C}_{tract}$ is, in effect, the union of two different classes: the first is the class of PDE settings that satisfy conditions (1) and (2.1), while the second is the class of PDE settings that satisfy conditions (1) and (2.2). The first of these classes can be described as the class of PDE settings $\mathcal{P} = (\mathbf{S},\mathbf{T},\Sigma_{st},\Sigma_{ts},\emptyset)$ in which every target-to-source tgd is has exactly one literal in its left-hand side which has no repeated variables. Hence, this is the class of PDE settings in which the target-to-source tgds are local-as-view (LAV) dependencies, an important class in data integration [15].

The second class contains as a subclass the family of all PDE settings $\mathcal{P} = (\mathbf{S},\mathbf{T},\Sigma_{st},\Sigma_{ts},\emptyset)$ in which every source-to-target tgd

is a *full tgd*, which means that it is of the form $\varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x})$. Indeed, if every source-to-target tgd is full, then the only marked variables are the ones that are existentially quantified in some target-to-source tgd. If two such variables appear together in the right-hand side of some target-to-source tgd D, then neither appears in the left-hand side of D, hence condition (2.2) (b) is satisfied.

We are now ready to state the main result of this section.

THEOREM 4. *Let $\mathcal{P}$ be a PDE setting in $\mathcal{C}_{tract}$. Then, $\mathsf{SOL}(\mathcal{P})$, the problem of testing for the existence of solutions is solvable in polynomial time.*

The proof of Theorem 4 uses properties of the chase procedure and homomorphism techniques. An outline of this proof will be given in the next section. In the remainder of this section, we derive some corollaries and then show that, in a certain sense, $\mathcal{C}_{tract}$ is a maximal class of tractable PDE settings.

COROLLARY 1. *Let $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \emptyset)$ be a PDE setting with no target constraints. If Σ_{st} is a set of full dependencies, then testing for the existence of solutions is solvable in polynomial time.*

COROLLARY 2. *Let $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \emptyset)$ be a PDE setting with no target constraints. If every target-to-source dependency of Σ_{ts} has exactly one literal on its left hand side which has no repeated variables, then testing for the existence of solutions is solvable in polynomial time.*

We now show that the conditions defining $\mathcal{C}_{tract}$ are tight, in the sense that minimal relaxations of them lead to intractability. Let us consider again the PDE setting used in the proof of Theorem 3, for which $\mathsf{SOL}(\mathcal{P})$ is NP-complete:

$$\begin{array}{ll} \Sigma_{st}: & D(x,y) \rightarrow \exists z \exists w P(x,z,y,w) \\ \Sigma_{ts}: & P(x,z,y,w) \rightarrow E(z,w) \\ & P(x,z,y,w) \wedge P(x,z',y',w') \rightarrow S(z,z') \end{array}$$

As seen earlier, the marked variables of Σ_{ts} are z and w (for the first tgd), and z, w, z', and w' (for the second tgd). Not surprisingly, this PDE setting does not belong to $\mathcal{C}_{tract}$, since it violates both condition (2.1) and condition (2.2) in Definition 9. These violations, however, are minimal. Indeed, condition (2.1) is violated because just one of the target-to-source tgds has two conjuncts in its left-hand side. Furthermore, condition (2.2) is violated because the marked variables z and z' appear in the only literal of the right-hand side of the second target-to-source tgd, but do not appear together in one of the conjuncts of the left-hand side; nonetheless, they are at distance two of each other, as they are "connected" via the variable x. Thus, the condition of being adjacent in the *Gaifman graph* of the variables in the left-hand side of the tgd cannot be relaxed to even being connected via a path of length two.

Next, we show that the intractability boundary is crossed if target constraints are allowed. In the following two PDE settings, the source-to-target and target-to-source constraints satisfy the conditions of $\mathcal{C}_{tract}$ and yet the existence-of-solutions problem is NP-hard for these settings.

Consider the following PDE setting:

$$\begin{array}{ll} \Sigma_{st}: & D(x,y) \rightarrow \exists z \exists w P(x,z,y,w) \\ \Sigma_{t}: & P(x,z,y,w) \wedge P(x,z',y',w') \rightarrow z = z' \\ \Sigma_{ts}: & P(x,z,y,w) \rightarrow E(z,w) \end{array}$$

The CLIQUE problem is reducible to the existence-of-solutions problem for this PDE setting, yet Σ_{st} and Σ_{ts} satisfy conditions (1) and (2.1) of Definition 9. Note that this setting contains a single target egd.

Next, consider the following PDE setting:

$$\begin{array}{ll} \Sigma_{st}: & S(z,w) \rightarrow S'(z,w) \\ & D(x,y) \rightarrow \exists z \exists w P(x,z,y,w) \\ \Sigma_{t}: & P(x,z,y,w) \wedge P(x,z',y',w') \rightarrow S'(z,z') \\ \Sigma_{ts}: & S'(z,z') \rightarrow S(z,z') \\ & P(x,z,y,w) \rightarrow E(z,w). \end{array}$$

Again, the CLIQUE problem is reducible to the existence-of-solutions problem p for this PDE setting, yet Σ_{st} and Σ_{ts} satisfy conditions (1) and (2.1) of Definition 9. Note that the target constraints contain a single full tgd.

Finally, we show that the intractability boundary is also crossed if we allow disjunctions in the right-hand side of target-to-source tgds. For this, consider the following PDE setting:

$$\begin{array}{ll} \Sigma_{st}: & E(x,y) \rightarrow \exists u C(x,u) \\ & E(x,y) \rightarrow E'(x,y) \\ \Sigma_{ts}: & E'(x,y) \wedge C(x,u) \wedge C(y,v) \rightarrow \\ & (R(u) \wedge B(v)) \vee (R(u) \wedge G(v)) \vee \\ & (B(u) \wedge G(v)) \vee (B(u) \wedge R(v)) \vee \\ & (G(u) \wedge R(v)) \vee (G(u) \wedge B(v)) \end{array}$$

The source relations are E, R, B, and G, while the target relations are E' and C. Given a graph E, we construct a source instance consisting of E, $R = \{r\}$, $G = \{g\}$ and $B = \{b\}$; we also take the target instance J to be empty. It easy to see that E is 3-colorable if and only if there is a solution for this PDE setting. Note that Σ_{st} and Σ_{ts} satisfy conditions (1) and (2.2) of Definition 9, and there are no target constraints.

5. Outline of the Proof of Theorem 4

In this section, we outline the proof of the tractability result presented in Theorem 4 of last section. We present an algorithm that decides the existence-of-solutions problem for PDE settings in the class $\mathcal{C}_{tract}$, and outline why it is a correct polynomial-time algorithm for this task.

The algorithm relies on the chase procedure and homomorphism techniques.[1] The chase procedure is used to construct a "representative" instance, which we call I_{can}, that can be used to decide the existence-of-solutions problem for a given (I, J) and a fixed PDE setting $\mathcal{P}$. The instance I_{can} is representative in the sense that we show that $\mathsf{SOL}(\mathcal{P})$ can be reduced to the problem of checking whether there is a homomorphism from I_{can} to I. Although the latter problem is NP-complete in general, we will prove that it is tractable when I_{can} is obtained by chasing the dependencies of a PDE setting in the class $\mathcal{C}_{tract}$.

The instance I_{can} is obtained by chasing the input instances (I, J) with the dependencies Σ_{st} and Σ_{ts} of the PDE setting (recall that Σ_t is empty in $\mathcal{C}_{tract}$). More precisely, let (I, J_{can}) be the result of chasing (I, J) with the source-to-target dependencies Σ_{st}. Then, I_{can} is a source instance such that (J_{can}, I_{can}) is the result of chasing $(J_{can}, \emptyset)$ with the target-to-source constraints Σ_{ts}. Notice that, since I_{can} is obtained by chasing tgds, it may contain null values.

[1] From now on, we will assume the definition of the chase procedure given in [9] (that is, the chase is no longer solution-aware).

The next theorem establishes the connection between $\mathsf{SOL}(\mathcal{P})$ and the problem of checking whether there is a homomorphism between I_{can} and I.

THEOREM 5. *Let $\mathcal{P}$ be a PDE setting such that for every tgd D in Σ_{ts}, every marked variable of D appears at most once in the left-hand side of D. Let I be a source instance, and J be a target instance. Let J_{can} be such that (I, J_{can}) is the result of chasing (I, J) with Σ_{st}. Let I_{can} be such that (J_{can}, I_{can}) is the result of chasing $(J_{can}, \emptyset)$ with Σ_{ts}. Then, there exists some solution for (I, J) in $\mathcal{P}$ iff there is some homomorphism from I_{can} to I.*

Before giving the proof of this theorem, we introduce some auxiliary results. The next lemma shows that, when there are no target-to-source dependencies, the result of chasing (I, J) with the source-to-target dependencies is an instance that has a homomorphism to every solution. The proof, which we omit for lack of space, is a straightforward adaptation of the proof of Theorem 3.3 of [9].

LEMMA 3. *Let $\mathcal{P}$ be a PDE setting where Σ_{st} consists of tgds, and Σ_t and Σ_{ts} are empty. Let I be a source instance (which may contain null values), and let J be a target instance. Let (I, J_{can}) be the result of chasing (I, J) with Σ_{st}. Then, there is a homomorphism from J_{can} to J_{sol}, for every solution J_{sol} for (I, J) in $\mathcal{P}$.*

The next lemma states that if there is a homomorphism between two instances K and K', and we chase them with a set of tgds to obtain instances L and L', then there is some homomorphism between L and L'. It follows easily from Lemma 3.4 of [9].

LEMMA 4. *Let Σ be a set of tgds. Let K and K' be instances (which may contain null values) such that there is a homomorphism from K to K'. Let L be the result of chasing K with Σ, and L' be the result of chasing K' with Σ. Then, there is a homomorphism from L to L'.*

In the proof of Theorem 5, we will construct an instance J_{img} that is a solution to the PDE setting. The instance J_{img} is the result of applying a homomorphism h to J_{can}. To show that J_{img} is a solution for the PDE setting, we rely on the following property which we shall show in Lemma 5: whenever a chase rule applies to a set of tuples of J_{img}, it also applies to the corresponding tuples of J_{can}. More precisely, let X be a set of tuples of J_{can} and Y be a set of tuples of J_{img} such that $h(X) = Y$. Whenever Y satisfies the left-hand side of a dependency D, so must X. It is easy to see that this property does not hold in general. For example, consider a tgd that maps paths of length two of the target to the source: $T_1(x, y) \wedge T_2(y, z) \rightarrow S(x, z)$. Let $X = \{T_1(A, B), T_2(C, D)\}$ be a set of tuples of J_{can} and let h be a homomorphism such that $h(A) = A$, $h(B) = B$, $h(C) = B$ and $h(D) = D$. Let $Y = h(X)$, that is $Y = \{T_1(A, B), T_2(B, D)\}$. Clearly, Y satisfies the left-hand side of the tgd, but X does not. Note that variable y appears in two literals of the tgd and the null values B and C appear at the positions of y in the tuples of X. It is easy to show that null values appear only at positions where there is a marked variable. Therefore, the variable y is a marked variable that appears twice in the left-hand side of the tgd. This, however, violates condition 1 of class $\mathcal{C}_{tract}$ (Definition 9). We show next that if condition 1 of $\mathcal{C}_{tract}$ is satisfied, we get the desired property.

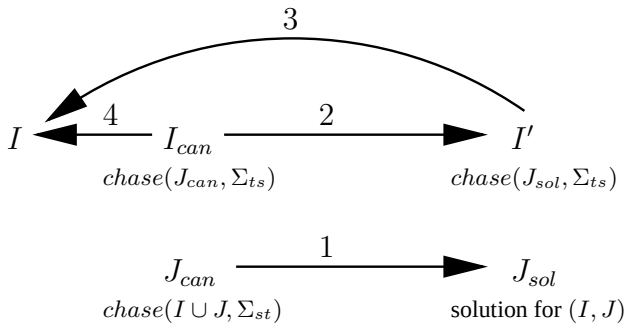

Figure 2: A diagram to illustrate Theorem 5

LEMMA 5. *Let $\mathcal{P}$ be a PDE setting such that $\mathcal{P}$ satisfies condition 1 of the definition of $\mathcal{C}_{tract}$. Consider a dependency of Σ_{ts} of the form $\forall \mathbf{x}\, \alpha_t(\mathbf{x}) \rightarrow \exists \mathbf{y}\, \beta_s(\mathbf{x}, \mathbf{y})$. Let I be a source instance, and J be a target instance. Let J_{can} be such that (I, J_{can}) is the result of chasing (I, J) with Σ_{st}. Let h be a function that preserves constants. Let $J_{img} = h(J_{can})$. Assume that there are tuples $T_1(\mathbf{c}_1), \ldots, T_m(\mathbf{c}_m)$ in J_{img} such that $T_1(\mathbf{c}_1), \ldots, T_m(\mathbf{c}_m) \models \alpha_t(\mathbf{x})$. Then, there are tuples $T_1(\mathbf{d}_1), \ldots, T_m(\mathbf{d}_m)$ in J_{can} such that $T_1(\mathbf{d}_1), \ldots, T_m(\mathbf{d}_m) \models \alpha_t(\mathbf{x})$, and $h(\mathbf{d}_i) = \mathbf{c}_i$ for $1 \leq i \leq m$.*

We are now ready to prove Theorem 5.

PROOF. ($\Rightarrow$) We will illustrate this direction of the proof with the diagram of Figure 2. Let J_{sol} be a solution for (I, J) in $\mathcal{P}$. Let I' be a source instance such that (J_{sol}, I') is the result of chasing $(J_{sol}, \emptyset)$ with Σ_{ts}. We will show that there is a homomorphism from I_{can} to I (arrow 4 in the diagram), by composing a homomorphism from I_{can} to I' (arrow 2) with a homomorphism from I' to I (arrow 3).

Recall that J_{can} is obtained by chasing only the dependencies of Σ_{st}. Thus, by Lemma 3 above, there is a homomorphism from J_{can} to every solution. In particular, since J_{sol} is a solution, there is a homomorphism from J_{can} to J_{sol} (arrow 1 of the diagram). By Lemma 4, there is a homomorphism from I_{can} to I' (arrow 2 of the diagram). Since (J_{sol}, I') is the result of chasing $(J_{sol}, \emptyset)$ with Σ_{ts} only, by Lemma 3, there is a homomorphism from I' to I (arrow 3 in the diagram).

($\Leftarrow$) Let h be a homomorphism from I_{can} to I. We shall construct an instance J_{img} and show that J_{img} is a solution for (I, J) in $\mathcal{P}$. We define J_{img} as the result of applying the following function h_J to J_{can}:

- $h_J(x) = h(x)$ if $x \in Dom(I_{can}) \cap Dom(J_{can})$
- $h_J(x) = x$ if $x \in Dom(J_{can}) - Dom(I_{can})$

where $Dom(I_{can})$ and $Dom(J_{can})$ denote the active domain of I_{can} and J_{can}, respectively. In order to show that J_{img} is a solution for (I, J) in $\mathcal{P}$, we will show that that $J \subseteq J_{img}$, $(I, J_{img}) \models \Sigma_{st}$, and $(J_{img}, I) \models \Sigma_{ts}$.

Since (I, J_{can}) is obtained by chasing (I, J) with Σ_{st}, we have that $J \subseteq J_{can}$. Since J is an instance without null values, and h_J preserves constants, $h_J(J) = J$. Therefore, $J \subseteq J_{img}$.

Algorithm `ExistsSolution`$_{\mathcal{P}}(I, J)$: *boolean*
Let J_{can} be such that (I, J_{can}) is the result of chasing (I, J) with Σ_{st}.
Let I_{can} be such that (J_{can}, I_{can}) is the result of chasing $(J_{can}, \emptyset)$ with Σ_{ts}.
for each block $I_{\mathcal{B}}$ of I_{can} **do**
if there is no homomorphism from $I_{\mathcal{B}}$ to I **then**
return `false`
end if
end for
return `true`

Figure 3: Algorithm `ExistsSolution`

Consider a tgd of Σ_{st} of the form $\forall \mathbf{x}.\phi_s(\mathbf{x}) \rightarrow \exists \mathbf{y}.\psi_t(\mathbf{x}, \mathbf{y})$. Assume that there is some $\mathbf{c}$ such that $I \models \phi_s(\mathbf{c})$. Notice that $\mathbf{c}$ is a vector of constants from $Dom(I)$, since I is an instance without null values. Since (I, J_{can}) is the result of chasing (I, J) with Σ_{st}, we have that $(I, J_{can}) \models \Sigma_{st}$. Therefore, $J_{can} \models \psi_t(\mathbf{c}, \mathbf{d})$, for some $\mathbf{d}$. Since $J_{img} = h_J(J_{can})$, h_J is a homomorphism from J_{can} to J_{img}. Since conjunctive queries are preserved under homomorphisms, $h_J(J_{can}) \models \psi_t(h_J(\mathbf{c}), h_J(\mathbf{d}))$. Since h_J preserves constants, $h_J(\mathbf{c}) = \mathbf{c}$. Thus, $J_{img} \models \psi_t(\mathbf{c}, \mathbf{e})$, for some $\mathbf{e}$. We conclude that $(I, J_{img}) \models \Sigma_{st}$.

Consider a tgd of Σ_{ts} of the form $\forall \mathbf{x}\ \alpha_t(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \beta_s(\mathbf{x}, \mathbf{y})$. Assume that there is some $\mathbf{c}$ in $Dom(J_{img})$ such that $J_{img} \models \alpha_t(\mathbf{c})$. By Lemma 5, it follows that $J_{can} \models \alpha_t(\mathbf{d})$, for some $\mathbf{d}$ in $Dom(J_{can})$ where $\mathbf{c} = h(\mathbf{d})$. Since I_{can} is obtained from the chase of $(J_{can}, \emptyset)$ with Σ_{ts}, we have that $(J_{can}, I_{can}) \models \Sigma_{ts}$. Thus, there is some $\mathbf{e}$ such that $I_{can} \models \beta_s(\mathbf{d}, \mathbf{e})$. Since h is a homomorphism from I_{can} to I, and conjunctive queries are preserved under homomorphisms, it is the case that $I \models \beta_s(h(\mathbf{d}), h(\mathbf{e}))$. Since $\mathbf{c} = h(\mathbf{d})$, we have that $I \models \beta_s(\mathbf{c}, \mathbf{f})$, for some $\mathbf{f}$. Therefore, $(J_{img}, I) \models \Sigma_{ts}$.

Since $J \subseteq J_{img}$, $(I, J_{img}) \models \Sigma_{st}$, and $(J_{img}, I) \models \Sigma_{ts}$, we conclude that J_{img} is a solution for (I, J) in $\mathcal{P}$. □

We now present the algorithm `ExistsSolution`$_{\mathcal{P}}(I, J)$ (shown in Figure 3) which decides whether there is a solution for (I, J) in the PDE setting $\mathcal{P}$. The algorithm first partitions I_{can} into a set of instances that we call *blocks*. Then, it checks whether there is a homomorphism from each block of I_{can} to I. The notion of *block* is adapted from [7] and defined as follows.

DEFINITION 10. Let K be an instance. The *graph of the nulls* of K is an undirected graph in which: (1) the nodes are all the nulls of K, and (2) there is an edge between two nulls whenever the nulls appear together in some tuple of K.

We say that $K_{\mathcal{B}}$ is a *block of tuples* of K if $K_{\mathcal{B}}$ is a maximal subset of K that satisfies one of the following conditions: (1) there exists a connected component $\mathcal{B}$ in the graph of the nulls of K such that every tuple of $K_{\mathcal{B}}$ has some null value from $\mathcal{B}$; or (2) there are no null values in $K_{\mathcal{B}}$.

The correctness of the algorithm follows from the next proposition and Theorem 5.

PROPOSITION 1. *There is a homomorphism from I_{can} to I if and only if there exists a homomorphism from $I_{\mathcal{B}}$ to I for every block $I_{\mathcal{B}}$ of I_{can}.*

In order to show that the algorithm runs in polynomial time for PDE settings of class $\mathcal{C}_{tract}$, we must prove that the problem of checking the existence of a homomorphism from each block of I_{can} to I is in P. We prove this by showing that every block of I_{can} has a constant number of null values. If there are source-to-target dependencies only, the result follows easily. Although the result still holds in the presence of target-to-source tgds, the proof is much more involved (Theorem 6 next). The polynomial running time of the algorithm follows from the fact that the problem of checking for the existence of a homomorphism from an instance with a constant number of null values to an arbitrary instance is tractable.

THEOREM 6. *Let $\mathcal{P}$ be a PDE setting that satisfies condition 2 of the definition of $\mathcal{C}_{tract}$. Let I be a source instance, and J be a target instance. Let J_{can} be such that (I, J_{can}) is the result of chasing (I, J) with Σ_{st}. Let I_{can} be such that (J_{can}, I_{can}) is the result of chasing $(J_{can}, \emptyset)$ with Σ_{ts}. Then, every block of tuples of I_{can} has a constant number of null values.*

Note that we only assume one of the two conditions of the definition of $\mathcal{C}_{tract}$ (condition 2). In turn, condition 2 is split into two subconditions: 2.1 and 2.2. The proof of Theorem 6 consists of two parts. In the first, we assume subcondition 2.1, and show that every block of I_{can} has a constant number of null values. In the second part, we do the same assuming subcondition 2.2.

The next lemma will be used in the first part of the proof of Theorem 6. It states that, assuming that the PDE satisfies subcondition 2.1, every block of I_{can} is the result of chasing exactly one block of J_{can}. The proof is by induction in the size of the blocks of I_{can}.

LEMMA 6. *Let $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \Sigma_t)$ be a PDE setting such that $\mathcal{P}$ satisfies condition 2.1 of the definition of $\mathcal{C}_{tract}$. Let I be a source instance, and J be a target instance. Let J_{can} be such that (I, J_{can}) is the result of chasing (I, J) with Σ_{st}. Let I_{can} be such that (J_{can}, I_{can}) is the result of chasing $(J_{can}, \emptyset)$ with Σ_{ts}. Let $I_{\mathcal{B}}$ be a block of I_{can}. Then, there exists a block of tuples $J_{\mathcal{B}}$ of J_{can} such that $I_{\mathcal{B}}$ is the result of chasing $J_{\mathcal{B}}$ with Σ_{ts}.*

PROOF. **Base case.** Assume that $I_{\mathcal{B}}$ has exactly one tuple $S(\mathbf{c})$. Since every dependency of Σ_{ts} has exactly one literal on the left-hand side, $S(\mathbf{c})$ is the result of chasing exactly one tuple of J_{can}.

Inductive step. Let $S(\mathbf{c})$ be a tuple of $I_{\mathcal{B}}$. Let $I'_{\mathcal{B}} = I_{\mathcal{B}} - \{S(\mathbf{c})\}$. By inductive hypothesis, every tuple of $I'_{\mathcal{B}}$ is in the result of chasing some block $J_{\mathcal{B}}$ of J_{can}. Since $I_{\mathcal{B}}$ is a block of tuples and $S(\mathbf{c}) \in I_{\mathcal{B}}$, there is some tuple $S'(\mathbf{c}')$ in $I'_{\mathcal{B}}$ such that $S(\mathbf{c})$ and $S'(\mathbf{c}')$ share some null value w.

Assume that w is a null value from $Var(J_{can})$. Since every dependency of Σ_{ts} has exactly one literal on the left-hand side, $S(\mathbf{c})$ is the result of chasing exactly one tuple $T(\mathbf{d})$ of J_{can}. Similarly, $S'(\mathbf{c}')$ is the result of chasing exactly one tuple $T'(\mathbf{d}')$ of J_{can}. Since w appears in $\mathbf{c}$ and $\mathbf{c}'$, and w is a null from J_{can}, w occurs in $\mathbf{d}$ and $\mathbf{d}'$. Since $S'(\mathbf{c}')$ is in $I'_{\mathcal{B}}$, $T'(\mathbf{d}')$ is in $J_{\mathcal{B}}$. Thus, $T(\mathbf{d})$ is also in $J_{\mathcal{B}}$. Consequently, $I_{\mathcal{B}}$ is the result of chasing $J_{\mathcal{B}}$ with Σ_{ts}.

Assume that w is a null value such that $w \notin Var(J_{can})$. Therefore, w is a null that is newly created during the chase of J_{can} with

Σ_{ts}. That is, w is created due to an existentially-quantified variable of a tgd of Σ_{ts}. Since every dependency of Σ_{ts} has exactly one literal on the left-hand side, $S(\mathbf{c})$ and $S'(\mathbf{c}')$ are in the result of chasing exactly one tuple $T(\mathbf{d})$ of J_{can}. Since $S'(\mathbf{c}')$ is in $I'_{\mathcal{B}}$, $T(\mathbf{d})$ is in $J_{\mathcal{B}}$. Consequently, $I_{\mathcal{B}}$ is the result of chasing $J_{\mathcal{B}}$ with Σ_{ts}. □

The next two lemmas will be used in the second part of the proof of Theorem 6 (i.e., assuming that the PDE satisfies subcondition 2.2). Recall that the null values of I_{can} may be created during the chase of the dependencies of either Σ_{st} or Σ_{ts}. All the null values that are created during the chase of dependencies of Σ_{st} appear in J_{can}. The following lemma states that, for every block $I_{\mathcal{B}}$ of I_{can}, the null values of $I_{\mathcal{B}}$ that were created during the chase of Σ_{st} not only appear in J_{can} but they also come from exactly one block of J_{can}. The proof is by induction in the size of the blocks of I_{can}.

LEMMA 7. *Let $\mathcal{P} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_{ts}, \Sigma_t)$ be a PDE setting such that $\mathcal{P}$ satisfies condition 2.2 of the definition of $\mathcal{C}_{tract}$. Let I be a source instance, and J be a target instance. Let J_{can} be such that (I, J_{can}) is the result of chasing (I, J) with Σ_{st}. Let I_{can} be such that (J_{can}, I_{can}) is the result of chasing $(J_{can}, \emptyset)$ with Σ_{ts}. Let $I_{\mathcal{B}}$ be a block of I_{can}. Then, there exists a block of tuples $J_{\mathcal{B}}$ of J_{can} such that for every null value w in $Var(I_{\mathcal{B}}) \cap Var(J_{can})$, $w \in Var(J_{\mathcal{B}})$.*

PROOF. **Base case.** Assume that $I_{\mathcal{B}}$ has exactly one tuple $S(\mathbf{c})$. Assume that there are null values w and z in $S(\mathbf{c})$ such that w and z appear in J_{can}. Let D be the dependency of Σ_{ts} such that $S(\mathbf{c})$ is the result of chasing some tuples $T_1(\mathbf{d}_1), \ldots, T_m(\mathbf{d}_m)$ of J_{can} with D. Let h be a homomorphism from $T_1(\mathbf{x}_1), \ldots, T_m(\mathbf{x}_m)$ to $T_1(\mathbf{d}_1), \ldots, T_m(\mathbf{d}_m)$. Let x_w and x_z be variables such that $h(x_w) = w$ and $h(x_z) = z$. Since there are null values at the position of x_w and x_z in J_{can}, x_w and x_z are marked variables in D. Since w and z appear in $S(\mathbf{c})$, x_w and x_z appear together in a literal of the right-hand side of D. Since D satisfies condition 2.2 of $\mathcal{C}_{tract}$, x_z and x_w appear together in some literal $T_i(\mathbf{x})$ of the left-hand side of D. Thus, w and z appear together in some tuple $T_i(\mathbf{d}_i)$ of J_{can}. Therefore, w and z belong to the same block of J_{can}.

Inductive step. Let $S(\mathbf{c})$ be a tuple of $I_{\mathcal{B}}$. Let $I'_{\mathcal{B}} = I_{\mathcal{B}} - \{S(\mathbf{c})\}$. By inductive hypothesis, there exists a block $J_{\mathcal{B}}$ of J_{can} such that every null value of $Var(I'_{\mathcal{B}}) \cap Var(J_{can})$ is in $Var(J_{\mathcal{B}})$. Assume that there is some null value in $S(\mathbf{c})$. By definition of block, there is some tuple $S'(\mathbf{c}')$ in $I'_{\mathcal{B}}$ such that $S(\mathbf{c})$ and $S'(\mathbf{c}')$ share some null value w.

Assume that w does not appear in J_{can}. Let D be the dependency of Σ_{ts} such that $S(\mathbf{c})$ is the result of chasing some tuples of J_{can} with D. Since w does not occur in J_{can}, it is at a position of $\mathbf{c}$ that corresponds to a marked variable which does not appear in the left-hand side of D (i.e., an existentially-quantified variable of D). Since $\mathcal{P}$ satisfies condition 2.2 of the definition of $\mathcal{C}_{tract}$, none of the nulls of $S(\mathbf{c})$ correspond to marked variables that appear on the left-hand side of D. Thus, none of the nulls of $S(\mathbf{c})$ are in J_{can}, and we are done.

Assume that w appears in J_{can}. Since w appears in $S'(\mathbf{c}')$, w is in $Var(J_{\mathcal{B}})$. Assume that there is some null value z from $Dom(J_{can})$ such that z occurs in $S(\mathbf{c})$ and z is distinct from w. We must prove now that z appears in $Var(J_{\mathcal{B}})$. Let D be the dependency of Σ_{ts} such that $S(\mathbf{c})$ is the result of chasing tuples $T_1(\mathbf{d}_1), \ldots, T_m(\mathbf{d}_m)$ of J_{can} with D, for some $\mathbf{d}_1, \ldots, \mathbf{d}_m$. Let h be a homomorphism from $T_1(\mathbf{x}_1), \ldots, T_m(\mathbf{x}_m)$ to $T_1(\mathbf{d}_1), \ldots, T_m(\mathbf{d}_m)$. Let x_w and x_z be variables such that $h(x_w) = w$ and $h(x_z) = z$. Since there are null values at the position of x_w and x_z in J_{can}, x_w and x_z are marked variables in D. Since w and z appear in $S(\mathbf{c})$, x_w and x_z appear together in a literal of the right-hand side of D. Since D satisfies condition 2.2 of $\mathcal{C}_{tract}$, x_z and x_w appear together in some literal $T_i(\mathbf{x})$ of the left-hand side of D. Thus, w and z appear together in some tuple $T_i(\mathbf{d}_i)$ of J_{can}. Since $w \in Var(J_{\mathcal{B}})$, $T_i(\mathbf{d}_i)$ is in $J_{\mathcal{B}}$. Thus, z appears in $J_{\mathcal{B}}$. □

The following lemma states that, if the PDE satisfies condition 2.2 of $\mathcal{C}_{tract}$, then the null values of each block come from the chase of either tgds of Σ_{st} or Σ_{ts}, but not both.

LEMMA 8. *Let $\mathcal{P}$ be a PDE setting such that $\mathcal{P}$ satisfies condition 2.2 of the definition of $\mathcal{C}_{tract}$. Let I be a source instance, and J be a target instance. Let J_{can} be such that (I, J_{can}) is the result of chasing (I, J) with Σ_{st}. Let I_{can} be such that (J_{can}, I_{can}) is the result of chasing $(J_{can}, \emptyset)$ with Σ_{ts}. Then, for every block of tuples $I_{\mathcal{B}}$ of I_{can}, exactly one of the following holds:*

- *all the null values of $I_{\mathcal{B}}$ are in J_{can}*
- *none of the null values of $I_{\mathcal{B}}$ are in J_{can}*

PROOF. Assume that some null value of $I_{\mathcal{B}}$ is from $Var(J_{can})$. Assume that $I_{\mathcal{B}}$ has some null values w and z such that $w \notin Var(J_{can})$ and $z \in Var(J_{can})$. By definition of block, there is a connected component $\mathcal{B}$ of the graph of the nulls of I_{can} such that z and w are nodes of $\mathcal{B}$. Thus, there are null values w' and z' in $\mathcal{B}$ such that $w' \notin Var(J_{can})$, $z' \in Var(J_{can})$, and w' and z' are adjacent in $\mathcal{B}$. Therefore, w' and z' appear together in some tuple $S(\mathbf{c})$ of I_{can}. Let D be the dependency of Σ_{ts} that, when chased, causes the addition of $S(\mathbf{c})$ to I_{can}. Since z' is in $Var(J_{can})$, it is at a position of $\mathbf{c}$ that corresponds to a marked variable that appears in the left-hand side of D. Since w' is not in J_{can}, it is at a position of $\mathbf{c}$ that corresponds to a marked variable which does not appear in the left-hand side of D (i.e., an existentially-quantified variable of D). Thus, $\mathcal{P}$ violates condition 2.2 of the definition of $\mathcal{C}_{tract}$; contradiction. □

We are now ready to prove Theorem 6. First, we claim that $J_{\mathcal{B}}$ has a constant number of tuples. Since I is an instance without null values, all the null values of $J_{\mathcal{B}}$ are created when chasing exactly one dependency of Σ_{st}. That is, there is a rule D of the form $\forall \mathbf{x}.\phi_s(\mathbf{x}) \rightarrow \exists \mathbf{y}.\psi_t(\mathbf{x}, \mathbf{y})$ such that all tuples of $J_{\mathcal{B}}$ are in $\psi_t(\mathbf{c}, \mathbf{d})$, for some $\mathbf{c}$ and $\mathbf{d}$. The size of ψ_t depends on the size of the dependency (which is assumed to be constant). Therefore, there is a constant number of tuples in $J_{\mathcal{B}}$.

Let $I_{\mathcal{B}}$ be a block of tuples of I_{can}. Assume that $\mathcal{P}$ satisfies condition 2.1 of the definition of $\mathcal{C}_{tract}$. By Lemma 6, there exists a block of tuples $J_{\mathcal{B}}$ of J_{can} such that $I_{\mathcal{B}}$ is the result of chasing $J_{\mathcal{B}}$ with Σ_{ts}. Since $J_{\mathcal{B}}$ has a constant number of tuples and $I_{\mathcal{B}}$ is the result of chasing $J_{\mathcal{B}}$ with Σ_{ts}, $I_{\mathcal{B}}$ has a constant number of tuples. Consequently, $I_{\mathcal{B}}$ has a constant number of null values.

Now, assume that $\mathcal{P}$ satisfies condition 2.2 of the definition of $\mathcal{C}_{tract}$. First, assume that none of the null values of $I_{\mathcal{B}}$ are from $Var(J_{can})$. Then, all the null values of $I_{\mathcal{B}}$ are created due to

existentially-quantified variables of dependencies of Σ_{ts}. Since each step of the chase creates new null values for the existentially-quantified variables, all the tuples of $I_{\mathcal{B}}$ are created when chasing exactly one dependency of Σ_{ts}. That is, there is a rule D of the form $\forall \mathbf{x}\ \alpha_t(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \beta_s(\mathbf{x}, \mathbf{y})$ such that all tuples of $I_{\mathcal{B}}$ are in $\beta_s(\mathbf{c}, \mathbf{d})$. The size of β_s depends on the size of the dependency (which is assumed to be constant). Therefore, there is a constant number of tuples in $I_{\mathcal{B}}$. Consequently, there is a constant number of null values in $I_{\mathcal{B}}$. Second, assume that $I_{\mathcal{B}}$ contains some null value from $Var(J_{can})$. Let N be the set of null values that appear in $I_{\mathcal{B}}$ and in J_{can}. By Lemma 7, there exists a block of tuples $J_{\mathcal{B}}$ of J_{can} such that every null value of N appears in $J_{\mathcal{B}}$. Since $J_{\mathcal{B}}$ has a constant number of tuples, N has a constant number of null values. Since $I_{\mathcal{B}}$ contains some null value from $Var(J_{can})$, by Lemma 8, $Var(I_{\mathcal{B}}) = N$.

6. Conclusions

We have introduced a framework for data sharing among independent peers which is a generalization of data exchange and a special case of peer data management. Peer data exchange models a scenario in which a target peer receives data from an autonomous source and has no authority to modify the data of the source peer. Nonetheless, the target peer may specify what data it is willing to receive, and the exchange makes use of source-to-target and target-to-source schema mappings. Within this conceptually simple yet powerful framework, we have shown that the existence-of-solutions problem is NP-complete. We have also explored the boundary between tractability and intractability, and identified a broad class of PDE settings for which the existence of solutions can be tested in polynomial time. We plan to further delineate this boundary and also investigate tractable extensions of $\mathcal{C}_{tract}$ that include target constraints.

We have also shown that the problem of obtaining certain answers in peer data exchange is coNP-complete for unions of conjunctive queries. This is in contrast to peer data management, where it is undecidable; and to data exchange, where it is tractable. We plan to investigate the complexity of computing certain answers for PDE settings in $\mathcal{C}_{tract}$ and to find classes of PDE settings with target constraints for which the problem of obtaining certain answers is tractable. Finally, we wish to explore alternative semantics when there is no solution. A semantics for query answering based on the semantics of repairs has been proposed [5]. However, the boundary between tractability and intractability for this semantics remains largely unexplored.

7. References

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *PODS*, pages 254–263, 1998.

[2] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[3] C. Beeri and M. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

[4] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for Peer-to-Peer computing: A vision. In *WebDB*, pages 89–94, 2002.

[5] L. Bertossi and L. Bravo. Query answering in peer-to-peer data exchange systems. In *EDBT Workshop on Peer-to-Peer Computing and Databases*, 2004.

[6] D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. Logical foundations of peer-to-peer data integration. In *PODS*, pages 241–251, 2004.

[7] R. Fagin, P. Kolaitis, and L. Popa. Data exchange: getting to the core. In *PODS*, pages 90–101, 2003. To appear in TODS.

[8] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.

[9] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *To appear in Theoretical Computer Science. In press.*, 2005.

[10] E. Franconi, G. Kuper, A. Lopatenko, and L. Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *VLDB Workshop on Databases, Information Systems and Peer-to-Peer Computing*, 2003.

[11] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. The coDB robust peer-to-peer database system. In *Symposium on Advanced Database Systems*, pages 382–393, 2004.

[12] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Spring-Verlag, Berlin, 1991. Lecture Notes in Computer Science, vol. 554.

[13] G. Grahne and A. Mendelzon. Tableau techniques for querying information sources through global schemas. In *ICDT*, pages 332–347, 1999.

[14] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *ICDE*, pages 505–518, 2003.

[15] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[16] C. Li. Raccoon: A peer-based system for data integration and sharing. In *ICDE*, page 852, 2004. System Demonstration.

[17] C. O'Donovan, M. J. Martin, A. Gattiker, E. Gasteiger, A. Bairoch, and R. Apweiler. High-quality protein knowledge resource: Swiss-prot and trembl. *Briefings in Bioinformatics*, 3(3):275–284, 2002.

[18] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *VLDB*, pages 598–609, 2002.

[19] I. Tatarinov. *Semantic Data Sharing with a Peer Data Management System*. PhD thesis, University of Washington, 2004.

[20] I. Tatarinov and A. Y. Halevy. Efficient query reformulation in peer data management systems. In *SIGMOD*, pages 539–550, 2004.

[21] R. van der Meyden. Logical approaches to incomplete information: A survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.