VLDB Journal manuscript No.
(will be inserted by the editor)

Schema Mediation for Large-Scale Semantic Data Sharing

Alon Y. Halevy[1], **Zachary G. Ives**[2], **Dan Suciu**[1], **Igor Tatarinov**[1]

[1] Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195-2350
e-mail: {alon,suciu,igor}@cs.washington.edu
[2] Department of Computer and Information Science, Moore School Building, University of Pennsylvania, 220 South 33rd Street, Philadelphia, PA 19104-6389
e-mail: zives@cis.upenn.edu

Received: date / Revised version: date

Abstract Intuitively, data management and data integration tools should be well-suited for exchanging information in a semantically meaningful way. Unfortunately, they suffer from two significant problems: they typically require a common and comprehensive schema design before they can be used to store or share information, and they are difficult to extend because schema evolution is heavyweight and may break backward compatibility. As a result, many large-scale data sharing tasks are more easily facilitated by non-database-oriented tools that have little support for semantics.

The goal of the peer data management system (PDMS) is to address this need: we propose the use of a decentralized, easily extensible data management architecture in which any user can contribute new data, schema information, or even mappings between other peers' schemas. PDMSs represent a natural step beyond data integration systems, replacing their single logical schema with an interlinked collection of semantic mappings between peers' individual schemas.

This paper considers the problem of schema mediation in a PDMS. Our first contribution is a flexible language for mediating between peer schemas, which extends known data integration formalisms to our more complex architecture. We precisely characterize the complexity of query answering for our language. Next, we describe a reformulation algorithm for our language that generalizes both global-as-view and local-as-view query answering algorithms. Then, we describe several methods for optimizing the reformulation algorithm, and an initial set of experiments studying its performance. Finally, we define and consider several *global* problems in managing semantic mappings in a PDMS.

Key words peer data management, data integration, schema mediation, web and databases

1 Introduction

While databases and data management tools excel at providing semantically rich data representations and expressive query languages, they have historically been hindered by a need for significant investment in design, administration, and schema evolution. Schemas must generally be predefined in comprehensive fashion, rather than evolving incrementally as new concepts are encountered; schema evolution is typically heavyweight and may "break" existing queries. As a result, many people find that database techniques are obstacles to lightweight data storage and sharing tasks, rather than facilitators. They resort to simpler and less expressive tools, ranging from spreadsheets to text files, to store and exchange their data. This provides a

simpler administrative environment (although some standardization of terminology and description is always necessary), but with a significant cost in functionality. Worse, when a lightweight repository grows larger and more complex in scale, there no easy migration path to a semantically richer tool.

Conversely, the strength of HTML and the World Wide Web has been easy and intuitive support for ad hoc extensibility — new pages can be authored, uploaded, and quickly linked to existing pages. However, as with flat files, the Web environment lacks rich semantics. That shortcoming spurred a movement towards XML, which allows data to be semantically tagged. Unfortunately, XML carries many of the same requirements and shortcomings as data management tools: for rich data to be shared among different groups, all concepts need to be placed into a common frame of reference. XML schemas must be completely standardized across groups, or mappings must be created between all pairs of related data sources.

Data integration systems have been proposed as a partial solution to this problem [GMPQ+97,HKWY97, ACPS96,LRO96,DG97,MFK01]. These systems support rich queries over large numbers of autonomous, heterogeneous data sources by exploiting the semantic relationships between the different sources' schemas. An administrator defines a global *mediated schema* for the application domain and specifies semantic mappings between sources and the mediated schema. We get the strong semantics needed by many applications, and data sources can evolve independently — and, it would appear, relatively flexibly. Yet in reality, the mediated schema, the integrated part of the system that actually facilitates all information sharing, becomes a bottleneck in the process. Mediated schema design must be done carefully and globally; data sources cannot change significantly or they might violate the mappings to the mediated schema; concepts can only be added to the mediated schema by the central administrator. The ad hoc extensibility of the web is missing, and as a result many natural, small-scale information sharing tasks are difficult to achieve.

We believe that there is a clear need for a new class of data sharing tools that preserves semantics and rich query languages, but which facilitates ad hoc, decentralized sharing and administration of data and defining of semantic relationships. Every participant in such an environment should be able to contribute new data and relate it to existing concepts and schemas, define new schemas that others can use as frames of reference for their queries, or define new relationships between existing schemas or data providers. We believe that a natural implementation of such a system will be based on a peer-to-peer architecture, and hence call such a system a *peer data management system* (PDMS). (We comment shortly on the differences between PDMSs and P2P file-sharing systems). The vision of a PDMS is to blend the extensibility of the HTML web with the semantics of data management applications. As we explain in a related paper [HITM03], peer-data management systems also provide an infrastructure on which to build applications for the Semantic Web [BLHL01].

Example 1 The extensibility of a PDMS can best be illustrated with a simple example. Figure 1 illustrates a peer data management system for supporting a web of database research-related data. This will be a running example thoughout the paper so we only describe the functionality here. Unlike a hierarchy of data

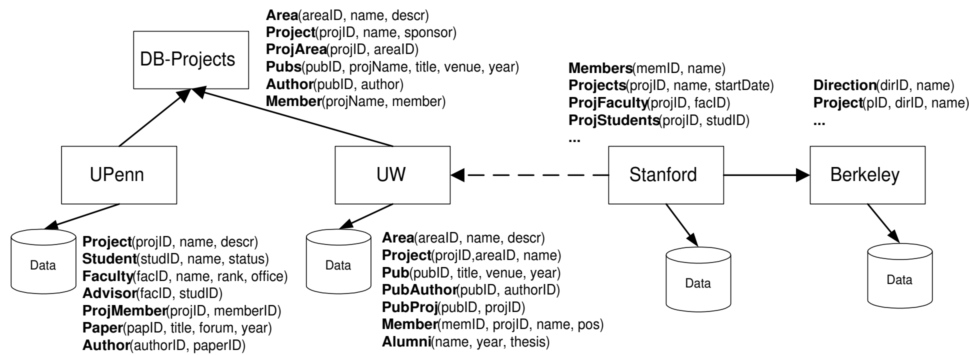

Fig. 1 A PDMS for the database research domain. Arrows indicate that there is (at least a partial) mapping between the relations of the peers. Only peer relations are shown; the stored relations at university peers are omitted. DB-Projects is a virtual, mediating peer that has no stored data. The figure illustrates how two semantic networks can be joined by establishing a single mapping between a pair of peers (UW and Stanford in this case).

integration systems or mediators, a PDMS supports any arbitrary network of relationships between peers. The true novelty lies in the PDMS's ability to exploit transitive relationships among peers' schemas. The figure shows that two semantic networks can be fully joined together with only a few mappings between similar members of each semantic network (in our example, we only required a single mapping). The new mapping from Stanford to UW enables any query at any of the five peers to access data at *all* other peers through transitive evaluation of semantic mappings. Importantly, we can add our mappings between the most similar nodes in the two semantic networks; this is typically much easier than attempting to map a large number of highly dissimilar schemas into a single mediated schema (as in conventional data integration).

It is important to emphasize that the ability to obtain relevant data from other nodes in the network depends on the existence of a *semantic path* to that node. The semantic path needs to relate the terms used in the query with the terms used by the node providing the data. Hence, it is likely that there will be information loss along long paths in the PDMS, because of missing (or incomplete) mappings, leading to the problem of how to *boost* a network of mappings in a PDMS. This paper considers only how to obtain the answers given a particular set of mappings, and also assumes the the mappings are correct, i.e., faithful to the data.

Our contributions: We are building the Piazza PDMS, whose goal is to support decentralized sharing and administration of data in the extensible fashion described above. Piazza investigates many of the logical, algorithmic, and implementation aspects of peer data management. In this paper, we focus strictly on the first issue that arises in such a system, namely the problem of providing decentralized schema mediation. In paricular, we focus on the topics of expressing mappings between schemas in such a system and answering queries over multiple schemas.

Research on data integration systems has provided a set of rich and well understood schema mediation languages upon which mediation in PDMSs can be built. The two commonly used formalisms are the *global-*

as-view (GAV) approach used by [GMPQ+97, HKWY97, ACPS96], in which the mediated schema is defined as a set of views over the data sources; and the *local-as-view* (LAV) approach of [LRO96, DG97, MFK01], in which the contents of data sources are described as views over the mediated schema. The semantics of the formalisms are defined in terms of *certain answers* to a query [AD98].

Porting these languages to the PDMS context poses two challenges. First, the languages are designed to specify relationships between a mediator and a set of data sources. In our context, they need to be modified to map between peers' schemas, where each peer can serve as both a data source and mediator. Second, the algorithms and complexity of query reformulation and answering in data integration are well understood for a *two-tiered* architecture. In the context of a PDMS, we would like to use the data integration languages to specify semantic relationships *locally* between small sets of peers, and answer queries *globally* on a network of semantically related peers. The key contributions of this paper are showing precisely when these languages can be used to specify local semantic relationships in a PDMS, and developing a query reformulation algorithm that uses local semantic relationships to answer queries in a PDMS.

We begin by describing a very flexible formalism, $\mathcal{PPL}$, (Peer-Programming Language, pronounced "people") for mediating between peer schemas, which uses the GAV and LAV formalisms to specify local mappings. We define the semantics of query answering for a PDMS by extending the notion of *certain answers* [AD98]. We present results that show the exact restrictions on $\mathcal{PPL}$ under which finding all the answers to the query can be done in polynomial time.

We then present a query reformulation algorithm for $\mathcal{PPL}$. Reformulation takes as input a peer's query and the formulas describing semantic relationships between peers, and it outputs a query that refers only to stored relations at the peers. Reformulation is challenging because peer mappings are specified locally, and answering a query may require piecing together multiple peer mappings to locate the relevant data. In a uniform fashion, our algorithm interleaves both global-as-view and local-as-view reformulation techniques. The algorithm is guaranteed to yield all the certain answers when they are possible to obtain. We describe several methods for optimizing the reformulation algorithm and describe an initial set of experiments whose goal is to test the performance bottlenecks of the algorithm.

Finally, a PDMS, being a network of semantic mappings, gives rise to several new problems as we consider the mappings from a *global* perspective. For example, we want to know when a semantic mapping is redundant, and we would like to compose semantic mappings in order to save in optimization time. These problems are especially important given that the set of mappings change when nodes join or leave the system. Here we consider a first fundamental problem underlying these issues, namely, when is it possible to say that two PDMSs are equivalent to each other? We define formally the problem of PDMS equivalence and prove that it is decidable in several interesting cases. The problem of *composing* semantic mappings is addressed in [HM03].

Before we proceed, we would like to emphasize the following points. First, this paper is not concerned with how semantic mappings are generated: this is an entire field of investigation in itself (see [RB01] for a recent

survey on schema mapping techniques). Second, while a PDMS is based on a peer-to-peer architecture, it is significantly different from a P2P file-sharing system (e.g., [Nap01]). In particular, joining a PDMS is inherently a more heavyweight operation than joining a P2P file-sharing system, since some semantic relationships need to be specified. Our initial architecture focuses on applications where peers are likely to stay available the majority of the time, but in which peers should be able to join (or add new data) very easily. We anticipate there will be a spectrum of PDMS applications, ranging from ad hoc sharing scenarios to ones in which the membership changes infrequently or is restricted due to security or consistency requirements.

The paper is organized as follows. Section 2 formally defines the peer mediation problem and describes our mediation formalism. Section 3 shows the conditions under which query answering can be done efficiently in our formalism. In Section 4 we describe a query reformulation algorithm for a PDMS, and Section 5 describes the results of our initial experiments. Section 6 discusses global PDMS considerations. Section 7 discusses related work and Section 8 concludes.

2 Problem Definition

In this section, we present the logical formalisms for describing a PDMS and the specification of semantic mappings between peers. Our goal is to leverage the techniques for specifying mappings in data integration systems, extending them beyond the two-tiered architecture.

In our discussion, for simplicity of exposition we assume the peers employ the relational data model, although in our implemented system peers share XML files and pose queries in a subset of XQuery that uses set-oriented semantics. Our discussion considers select-project-join queries with set semantics, and we use the notation of conjunctive queries. In this notation, joins are specified by multiple occurrences of the same variable. Unless explicitly specified, we assume queries do not contain comparison predicates (e.g., $\neq$, $<$). Views refer to named queries.

We assume that each peer defines its own relational *peer schema* whose relations are called *peer relations*; a query in a PDMS will be posed over the relations from a specific peer schema. Without loss of generality we assume that relation and attribute names are unique to each peer.

Peers may also contribute data to the system, in the form of *stored relations*. Stored relations are analogous to data sources in a data integration system: all queries in a PDMS will be reformulated strictly in terms of stored relations that may be stored locally or on other peers. (Note that not every peer needs to contribute stored relations to the system, as some peers may strictly serve as logical mediators to other peers.) We assume that the names of stored relations are distinct from those of peer relations. We will refer to the set of stored relations at a peer as the peer's *stored schema*.

Example 2 In our example PDMS in Figure 1, only peer relations are shown. The lines between peers indicate that there is a mapping (described later) between the relations of the two peers.

Stored relations containing actual data are provided by the universities: the UPenn, UW, Stanford, and Berkeley peers. DB-Projects is a separate peer that provides a uniform view over the UPenn and UW data. Stanford and Berkeley, as neighboring universities, came to an agreement to map their schemas directly. The flexibility of the PDBMS (due to its ability to evaluate transitive relationships between schemas) becomes evident when two PDMSs are joined. In our example, once a mapping between the Stanford-Berkeley PDMS and the UPenn-UW-DB-Projects PDMS is established, queries over any of the five peers will be able to access *all* of the stored relations.

Note that our approach can support evolving schemas very naturally. A new schema version can be treated as an additional peer schema. In general, the new version is likely to be very similar to the previous version making the problem of specifying a mapping between the versions rather easy. In addition, the resulting mapping is likely to be very accurate.

2.1 System Architecture

A Piazza PDMS consists of a set of network nodes (physical peers) connected to the Internet. Every peer node is associated with a peer schema and, optionally, a stored schema. In addition, a peer can define schema mappings as described in the following section. All this metadata information is stored in the Piazza catalog. We assume that the catalog is accessible to all of the peers in a PDMS, which can be achieved by either storing it centrally, at a designated peer or by employing any of the distributed hash index techniques.

2.2 A Mapping Language for PDMSs

Obviously, the power of the PDMS lies in its ability to exploit semantic mappings between peer and stored relations. In particular, there are two types of mappings that must be considered: (1) mappings describing the data within the stored relations (generally with respect to one or more peer relations), and (2) mappings between the schemas of the peers. At this point it is instructive to recall the formalisms used in the context of data integration systems, since we build upon them in defining our mapping description language.

2.2.1 Mappings in Data Integration Data integration systems provide a uniform interface to a multitude of data sources through a logical, virtual *mediated* schema. The mediated schema is virtual in the sense that it is used for posing queries, but not for storing data. Mappings are established between the mediated schema and the relations at the data sources, forming a two-tier architecture in which queries are posed over the mediated schema and evaluated over the underlying source relations (stored relations, in our terminology) . A data integration system can be viewed as a special case of a PDMS.

Two main formalisms have been proposed for schema mediation in data integration systems. In the first, called *global-as-view* (GAV) [SBD+81, GMPQ+97, HKWY97, ACPS96], the relations in the mediated schema

are defined as views over the relations in the sources. In the second, called *local-as-view* (LAV) [LRO96, DG97,MFK01,LKG99,FW97], the relations in the sources are specified as views over the mediated schema. In fact, in many cases the source relations are said to be *contained* in a view over the mediated schema, as opposed to being exactly equal to it. We illustrate both below.

Example 3 The DB-Projects' Member peer relation, which mediates UPenn and UW peers, may be expressed using a GAV-like definition. The definition specifies that Member in DB-Projects is obtained by a union over the UPenn and UW schemas. Note in our examples, that peer relations are named using a peer-name:relation-name syntax:

$$
\begin{aligned}
\mathsf{DBProjects:Member(projName,member)} :- \ & \mathsf{UPenn:Student(sid,member,_), UPenn:ProjMember(pid,sid),} \\
& \mathsf{UPenn:Project(pid,projName,_)} \\
\mathsf{DBProjects:Member(projName,member)} :- \ & \mathsf{UPenn:Faculty(sid,member,_), UPenn:ProjMember(pid,sid),} \\
& \mathsf{UPenn:Project(pid,projName,_)} \\
\mathsf{DBProjects:Member(projName,member)} :- \ & \mathsf{UW:Member(mid,pid,member,_), UW:Project(pid,_,projName)}
\end{aligned}
$$

We may use the LAV formalism to specify the UW peer relations as views over mediated DB-Projects relations. This formalism is especially useful when there are many data sources that are related to a particular mediated schema. In such cases, it is more convenient to describe the data sources as views over the mediated schema rather than the other way around. In our scenario, DB-Projects may eventually mediate between many universities, and hence LAV is appropriate for future extensibility. The following illustrates an LAV mapping for UW:

$$
\begin{aligned}
\mathsf{UW:Project(projID,areaID,projName)} \subseteq \ & \mathsf{DBProjects:Project(projID,projName),} \\
& \mathsf{DBProjects:ProjArea(projID,areaID)}
\end{aligned}
$$

The fundamental difference between the two formalisms is that GAV specifies how to extract tuples for the mediated schema relations from the sources, and hence query answering amounts to view unfolding. In contrast, LAV is source-centric, describing the contents of the data sources. Query answering requires algorithms for answering queries using views [Hal01], but in exchange LAV provides greater extensibility: the addition of new sources is less likely to require a change to the mediated schema.

Our goal in $\mathcal{PPL}$ is to preserve the features of both the GAV and LAV formalisms, but to extend them from a two-tiered architecture to our more general network of interrelated peer and stored relations. Semantic relationships in a PDMS will be specified between pairs (or small sets) of peer (and optionally stored) relations. Ultimately, a query over a given peer relation may be reformulated over stored relations on any peer in the transitive closure of peer mappings.

2.2.2 Mappings for PDMSs We now present the $\mathcal{PPL}$ language, which uses the data integration formalisms locally. First we formally define our two types of mappings, which we refer to as storage descriptions and peer mappings.

Storage descriptions: Each peer contains a (possibly empty) set of *storage descriptions* that specify which data it actually stores by relating its stored relations to one or more peer relations. Formally, a storage description of the form $A : R = Q$, where Q is a query over the schema of peer A and R is a stored relation at the peer. The description specifies that A stores in relation R the result of the query Q over its schema.

In many cases the data that is stored is not *exactly* the definition of the view, but only a subset of it. As in the context of data integration, this situation arises often when the data at the peer may be incomplete (this is often called the *open-world assumption* [AD98]).[1] Hence, we also allow storage descriptions of the form $A : R \subseteq Q$. Thus, storage descriptions can be of two kinds *containment* (or *inclusion) storage descriptions* and *equality storage descriptions.*

Example 4 A storage description might relate the stored students relation at peer UPenn to the peer relations:

$$\begin{aligned} \mathsf{UPenn : students(sid, name, advisor)} \subseteq\ & \mathsf{UPenn : Student(sid, name, _), UPenn : Advisor(sid, fid),} \\ & \mathsf{UPenn : Faculty(fid, advisor, _, _)} \end{aligned}$$

This storage description says that UPenn:students stores a subset of the join of Student, Advisor and Faculty, which reflects the fact that UPenn:students is unlikely to contain information about *all* students in the world; it will probably contain data on "local" students only. Hence, if a UPenn:Affiliation peer relation with the corresponding semantics was available, the above storage description could be specified more precisely as follows:

$$\begin{aligned} \mathsf{UPenn : students(sid, name, advisor)} =\ & \mathsf{UPenn : Student(sid, name, _), UPenn : Advisor(sid, fid),} \\ & \mathsf{UPenn : Faculty(fid, advisor, _, _), UPenn : Affiliation(sid, 'UPenn')} \end{aligned}$$

Peer mappings: *Peer mappings* provide semantic glue between the schemas of different peers. We have two types of peer mappings in $\mathcal{PPL}$. The first are *inclusion* and *equality* mappings (similar to the concepts for storage descriptions). In the most general case, these mappings are of the form $Q_1(\bar{\mathcal{A}}_1) = Q_2(\bar{\mathcal{A}}_2)$, (or $Q_1(\bar{\mathcal{A}}_1) \subseteq Q_2(\bar{\mathcal{A}}_2)$ for inclusions) where Q_1 and Q_2 are conjunctive queries with the same arity and $\bar{\mathcal{A}}_1$ and $\bar{\mathcal{A}}_2$ are *sets* of peers. Query Q_1 (Q_2) can refer to any of the peer relation in $\bar{\mathcal{A}}_1$ ($\bar{\mathcal{A}}_2$, resp.). Intuitively, such a statement specifies a semantic mapping by stating that evaluating Q_1 over the peers $\bar{\mathcal{A}}_1$ will always produce the same answer (or a subset in the case of inclusions) as evaluating Q_2 over $\bar{\mathcal{A}}_2$. Note that since $\mathcal{PPL}$ allows queries on both sides of the equation, they can accommodate both GAV and LAV-style mappings (and thus we can express any of the mappings from Section 2.2.1).

The second kind of peer mappings are called *definitional mappings.* A definitional mapping is a datalog rule whose head and body are both peer relations, i.e., the body cannot contain a query. Formally, as long as a peer relation appears only once in the head of a definitional description, such mappings can be written as equalities. We include definitional mappings in order to obtain the full power of GAV mappings. We distinguish definitional mappings for the following reasons:

- As we show in Section 3, the complexity of answering queries when equality mappings are restricted to being definitional is more attractive than the general case, and

[1] Sometimes it may be possible to describe the exact contents of a data source with a more refined query, but very often this cannot be done.

– Definitional mappings can easily express disjunction: e.g., $P(x) :- P_1(x)$ and $P(x) :- P_2(x)$ means that P is the union of P_1 and P_2 (while the pair of mappings $P(x) = P_1(x)$ and $P(x) = P_2(x)$ means that P, P_1 and P_2 are equal).

In summary, a PDMS N is specified by a set of peers $\{P_1, ..., P_n\}$, a set of peer schemas $\{S_1, ..., S_m\}$ and a mapping function from peers to schemas, a set of stored relations $\mathcal{R}_i$ at each peer P_i, a set of peer mappings $\mathcal{L}_N$, and a set of storage descriptions $\mathcal{D}_N$. The storage descriptions and peer mappings provided by a peer P_i may reference stored or peer relations defined by other peers, so any peer can extend another peer's relations or use its data.

2.3 Semantics of $\mathcal{PPL}$

Given the peer and stored relations, their mappings, and a query over some peer schema, the PDMS needs to answer the query using the data from the stored relations. To formally specify the problem of query answering, we need to define the semantics of queries. We show below how the notion of *certain answers* [AD98] from the data integration context can be generalized to our context. Our goal is to formally define what is the set of answers to a query Q posed over the relations of a peer A. The challenge arises because the peer schemas are virtual; in fact, some data may only exist partially, if at all, in the system.

Formally, we assume that we are given a PDMS N and an instance for the stored relations, D, i.e., a set of tuples $D(R)$ for each stored relation $R \in (\mathcal{R}_1 \cup \ldots \cup \mathcal{R}_n)$. A *data instance* I for a PDMS N is an assignment of a set of tuples to each peer relation in every peer. We denote by $I(R)$ the set of tuples assigned to the relation R by I, and we denote by $Q(I)$ the result of computing the query Q over the extensional data in I. To define certain answers, we will consider only the data instances that are consistent with the specification of N:

Definition 1 *Consistent data instance. A data instance I is said to be* consistent *with a PDMS N and an instance D for N's stored relations if:*

- *For every storage description in $\mathcal{D}_N$, if it is of the form $A : R = Q_1$ ($A : R \subseteq Q_1$), then $D(R) = Q_1(I)$ ($D(R) \subseteq Q_1(I)$).*
- *For every peer description in $\mathcal{L}_N$:*
  - *if it is of the form $Q_1(\mathcal{A}_1) = Q_2(\mathcal{A}_2)$, then $Q_1(I) = Q_2(I)$,*
  - *if it is of the form $Q_1(\mathcal{A}_1) \subseteq Q_2(\mathcal{A}_2)$, then $Q_1(I) \subseteq Q_2(I)$,*
  - *if it is a definitional description whose head predicate is p, then let $r_1, \ldots, r_m$ be all the definitional mappings with p in the head, and let $I(r_i)$ be the result of evaluating the body of r_i on the instance I. Then, $I(p) = I(r_1) \cup \ldots \cup I(r_m)$.*

Intuitively, a data instance I is consistent with N and D if it describes *one* possible state of the world (i.e., extension for each of the peer relations) that is allowable given the data and peer mappings and D. We define the certain answers to be those that hold in *every* possible consistent data instance:

Definition 2 *Certain answers. Let Q be a query over the schema of a peer A in a PDMS N, and let D be an instance of the stored relations of N. A tuple $\bar{a}$ is a* certain answer *to Q if $\bar{a}$ is in $Q(I)$ for every data instance that is consistent with N and D.*

Note that in the last bullet of Definition 1 we did not require that the extension of p be the least-fixed point model of the datalog rules. However, since we defined certain answers to be those that hold for *every* consistent data instance, we actually do get the intuitive semantics of datalog for these mappings.

Query answering: Now we can define the query answering problem: given a PDMS N, an instance of the stored relations D and a query Q, find all certain answers of Q.

Advantages of a PDMS

Before we proceed, we summarize the advantages of a PDMS as an architecture for data sharing. First, we note that a PDMS offers a generic architecture for sharing data. Data integration systems and their variants are an instance of the PDMS architecture. An important advantage of PDMS is that they facilitate ad-hoc extensions. When a data source wants to join the system, it can do so locally, by providing a semantic mapping to one or few existing data sources. These data sources can be chosen to be the ones most convenient for specifying semantic mappings, rather than a global or mediated schema. Finally, querying is easier in a PDMS – queries at a peer can be posed using the schema of the peer, which is already familiar to the user.

Section 3 considers the computational complexity of query answering, and section 4 describes an algorithm for finding all the certain answers.

3 Complexity of Query Answering

This section establishes the basic results on the complexity of finding the certain answers in a PDMS. The complexity will depend on the restrictions we impose on peer mappings in $\mathcal{PPL}$. The computational complexity of finding all certain answers is well understood for the data integration context with a two-tiered architecture of a mediator and a set of data sources [AD98]. The key contribution of this section is to show the complexity of query answering in the global context of a PDMS, when the data integration formalisms are used locally.

The focus of our analysis is on data complexity — the complexity of query answering in terms of the total size of the data stored in the peers. Typically, the complexity of query answering is either polynomial, Co-NP-hard but decidable, or undecidable. In the polynomial case, it is often possible to find a *reformulation*

of the query into a query that refers only to the stored relations. The reformulated query is then further optimized and then executed. In the latter two cases, it is not possible to find *all* certain answers efficiently; but it is possible to develop an efficient reformulation algorithm that does not provide all certain answers, but which *only* returns certain answers.

A basic result: We begin by showing that cyclicity of peer mappings plays a significant role in the complexity of answering queries.

Definition 3 *Acyclic inclusion peer mappings. A set $\mathcal{L}$ of inclusion peer mappings in $\mathcal{PPL}$, is said to be acyclic if the following directed graph is acyclic. The graph contains a node for every peer relation mentioned in $\mathcal{L}$. There is an arc from the node corresponding to R to the node corresponding to S if there is a peer description in $\mathcal{L}$ of the form $Q_1(\bar{A}_1) \subseteq Q_2(\bar{A}_2)$ where R appears in Q_1 and S appears in Q_2.*

The following theorem characterizes two extreme cases of query answering in a PDMS:

Theorem 1 *Let N be a PDMS specified in $\mathcal{PPL}$.*

1. *The problem of finding all certain answers to a conjunctive query Q, for a given PDMS N, is undecidable.*
2. *If N includes only inclusion peer and storage descriptions and the peer mappings are acyclic, then a conjunctive query can be answered in polynomial time data complexity.*

The difference in complexity between the first and second bullets shows that the presence of cycles is the culprit for achieving query answerability in a PDMS (note that equalities automatically create cycles). In a sense the theorem also establishes a limit on the arbitrary combination of the formalisms of LAV and GAV. The proof is based on a reduction from the implication problem for functional and inclusion dependencies ([AHV95], Theorem 9.2.4). The proof is given in the appendix.

The second bullet points out a powerful schema mediation language for PDMS for which query answering can be done efficiently. It shows that LAV and GAV style reformulations can be chained together arbitrarily, and extends the results of [FLM99], which combined one level of LAV followed by one level of GAV.

Cyclic PDMSs: Acyclic PDMSs may be too restrictive for practical applications. One particular case of interest is *data replication*: when one peer maintains a copy of the data stored at a different peer. For example, referring to Fig. 1, the UPenn peer may wish to replicate UW's publication data:

$$\mathsf{UPenn} : \mathsf{pubs}(\mathsf{p}, \mathsf{t}, \mathsf{v}, \mathsf{y}) = \mathsf{UW} : \mathsf{pubs}(\mathsf{p}, \mathsf{t}, \mathsf{v}, \mathsf{y})$$

This example illustrates that we need equality in order to express data replication, which introduces a cyclic PDMS (the two relations mutually include each other's contents). While in general query answering is undecidable, it becomes decidable when equalities are projection-free, as in this example. The following theorem shows an important special case where query answering is tractable, and two additional cases where it is decidable.

Theorem 2 *Let N be a PDMS for which all inclusion peer mappings are acyclic, but which may also contain equality peer mappings.*

1. *If the following two conditions hold: (1) whenever a storage or peer description in N is an equality description, it does not contain projections, and (2) a peer relation that appears in the head of a definitional description does not appear on the right-hand side of any other description, then the query answering problem is in polynomial time.*
2. *If condition (2) of the first bullet holds but condition (1) doesn't, i.e., some equality storage descriptions contain projections, then the data complexity of the query answering problem is co-NP complete.*
3. *If condition (1) of the first bullet holds but condition (2) doesn't, i.e., some of the queries on the right-hand side of the peer mappings contain disjunction, the data complexity of query answering is co-NP complete.*

Note that the first bullet in the theorem allows definitional mappings to be disjunctive, if there are multiple mappings with the same head predicate. The conditions of the first bullet describe the most relaxed conditions under which query answering is tractable, and extends the results of [AD98] for purely LAV mappings. In fact, the proof is an extension of the proofs given in [AD98] to the PDMS context. The algorithm described in the next section will find all the certain answers under these conditions. The two subsequent bullets show that relaxing the conditions of the first bullet cause the query answering problem to be intractable.

Adding comparison predicates: Many applications will make extensive use of comparison predicates in peer mappings. Comparison predicates are especially useful when many peers model the same type of data, but they are distinguished on ranges of certain values of attributes (e.g., author names, years of publication, price ranges, geographic location). The following theorem shows what happens when comparison predicates are introduced into the peer mappings of a PDMS. We note that the algorithm we describe in the next section finds all the certain answers when the PDMS satisfies the conditions of the first bullet.

Theorem 3 *Let N be a PDMS satisfying the same conditions as the first bullet of Theorem 2, and let Q be a conjunctive query.*

1. *If comparison predicates appear only in storage descriptions or in the bodies of definitional mappings, but not in Q, then query answering is in polynomial time.*
2. *Otherwise, the query answering problem is co-NP complete.*

The crux of the proof of the first bullet is based on the observations that the complexity results on finding certain answers in the LAV setting [LRO96] carry over to the case where the views contain comparison predicates, but the query does not. The second bullet follows from the hardness result in [AD98] for the LAV setting.

PDMS consistency

We say that a PDMS N and an assignment D to the stored relations are said to be inconsistent if there is no consistent data instance w.r.t. N and D. A natural question that arises is whether the mappings in a PDMS can entail that a PDMS has inconsistent data. As it turns out, the following theorem shows that inconsistency is possible only if we introduce equalities in storage descriptions.

Theorem 4 *Let N be a PDMS specified in $\mathcal{PPL}$. If all storage descriptions in N are inclusions, then for any assignment D to its stored relations there is always a consistent data instance w.r.t. N and D. If storage descriptions contain equalities, it may be possible to construct an assignment D to the stored relations such that there is no consistent data instance w.r.t. N and D.*

The theorem also sheds some light on the complexity of the query answering problem. When the PDMS may be in an inconsistent state, it is because the relationships between the stored relations are more subtle, and hence answering queries is also more expensive. When inconsistency is not possible, we can build a *minimal canonical instance* of the data in the PDMS efficiently, and all the certain answers can be obtained from that instance.

Summary: with arbitrary use of the data integration formalisms in a PDMS, query answering is undecidable. However, this section has shown that there is a powerful subset of $\mathcal{PPL}$ in which query answering is tractable. The subset allows both the LAV and GAV mediation languages, and it supports a limited form of cycles in the peer mappings and as well as limited use of comparison predicates. Hence, we obtain a flexible language for mediating between PDMS peers.

4 Query Reformulation Algorithm

In this section we describe an algorithm for query reformulation for PDMSs. The input of the algorithm is a set of peer mappings and storage descriptions and a query Q. The output of the algorithm is a query expression Q' that *only* refers to stored relations at the peers. To answer Q we need to evaluate Q' over the stored relations. The precise method of evaluating Q' is beyond the scope of this paper, but we note that recent techniques for adaptive query processing [IHW01] are well suited for our context.

The algorithm is sound and complete in the following sense. Evaluating Q' will always *only* produce certain answers to Q. When all the certain answers can be found in polynomial time (according to Section 3), Q' will produce all certain answers.

4.1 Algorithm overview

Before we describe the details of the algorithm, we first provide some intuition on its working and the challenges it faces. Consider a PDMS in which all peer mappings are definitional (similar to GAV mappings

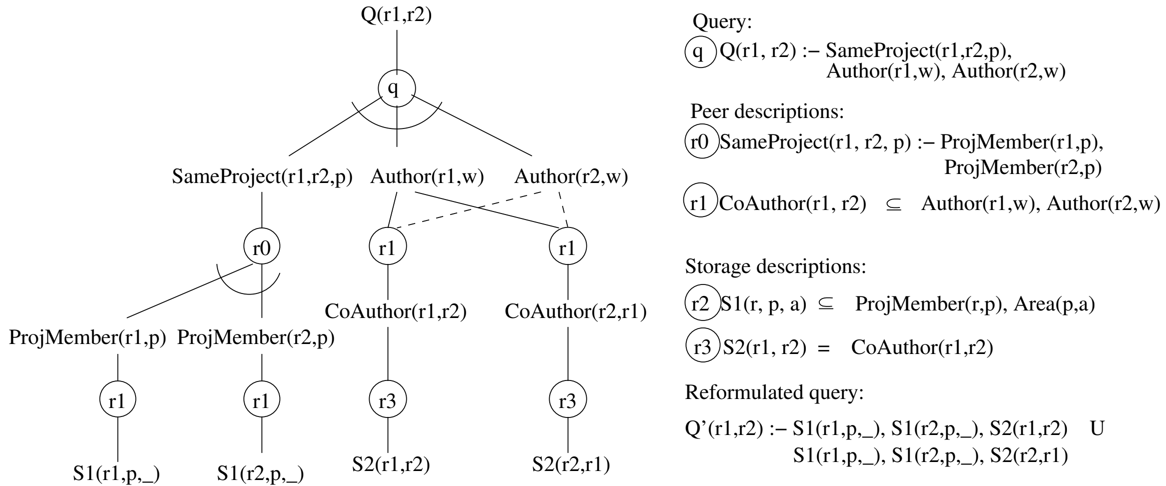

Fig. 2 Reformulation rule-goal tree for the database reseearch domain. Dashed lines represent nodes that are included in the *unc* label (see text).

in data integration). In this case, the algorithm is a simple construction of a rule-goal tree: goal nodes are labeled with atoms of the peer relations, and rule nodes are labeled with peer mappings. We begin by expanding each query subgoal according to the relevant definitional peer mappings in the PDMS. When none of the leaves of the tree can be expanded any further, we use the storage descriptions for the final step of reformulation in terms of the stored relations.

At the other extreme, suppose all peer mappings in the PDMS are inclusions in which the left-hand side has a single atom (similar to LAV mappings in data integration). In this case, we begin with the query subgoals and apply an algorithm for answering queries using views (e.g., [Hal01]). We apply the algorithm to the result until we cannot proceed further, and as in the previous case, we use the storage descriptions for the last step of reformulation.

The first challenge of the complete algorithm is to combine and interleave the two types of reformulation techniques. One type of reformulation replaces a subgoal with a set of subgoals, while the other replaces a set of subgoals with a single subgoal. The algorithm will achieve this by building a rule-goal tree, while it simultaneously marks certain nodes as covering not only their parent node, but also their uncle nodes (as described in the example below).

Example 5 To illustrate the rule-goal tree[2], Figure 2 shows an example for a simple query. We begin with the query, Q, which asks for researchers who have worked on the same project and also co-authored a paper. Q is expanded into its three subgoals, each of which appears as a goal node. The SameProject peer relation (indicating which researchers work on the same project) is involved in a single definitional peer description (r_0), hence we expand the SameProject goal node with the rule r_0, and its children are two goal nodes of the ProjMember peer relation (each specifying the projects an individual researcher is involved in).

[2] More precisely, we actually build a rule-goal DAG, as illustrated in the example.

The Author relation is involved in an inclusion peer description (r_1). We expand Author(r1,w) with the rule node r_1, and its child becomes a goal node of the relation CoAuthor. This "expansion" is of different nature because of the LAV-style reformulation. Intuitively, we are reformulating the Author(r1,w) subgoal to use the left-hand side of r_1. The right-hand side of r_1 includes two subgoals of Author (with the appropriate variable patterns), so we also mark r_1 as covering its *uncle* node. (In the figure, this annotation is indicated by a dashed line.) Since the peer relation Author is involved in a single peer description, we do not need to expand the subgoal Author(r2,w) any further. Note, however, that we must apply description r_1 a second time with the head variables reversed, since CoAuthor may not be symmetric (because it is $\subseteq$ rather than $=$).

At this point, since we cannot reformulate the peer mappings any further, we consider the storage descriptions. We find stored relations for each of the peer relations in the tree (S_1 and S_2), and produce the final reformulation. Reformulations of peer relations into stored relations can also be either in GAV or LAV style. In this simple example, our reformulation involves only one level of peer mappings, but in general, the tree may be arbitrarily deep.

The second challenge we face is that the rule-goal tree may be huge. First, the tree may be very deep, because it may need to follow any path through semantically related peers. Second, the branching factor of the tree may be large because data is replicated at many peers. Hence, it is crucial that we develop effective methods for pruning the tree and for generating first solutions quickly. It is important to emphasize that while many sophisticated methods have been developed for constructing rule-goal trees in the contexts of datalog analysis (e.g., [HMSS01,SR92]) and static analysis of logic programs [BDSK89], the focus in these works has been developing termination criteria that provide certain guarantees, rather than optimizing the construction of the tree itself.

Before proceeding, we recall the main aspect of algorithms for rewriting queries using views [PH01] that is germane to our discussion. Suppose we have the following query Q and views (we use the terminology of [PH01]):

$$\begin{array}{ll} Q(X,Y) & :-\ e_1(X,Z), e_2(Z,Y), e_3(X,Y) \\ V_1(A,B) & :-\ e_1(A,C), e_2(C,B) \\ V_2(D,E) & :-\ e_3(X,Y), e_4(Y) \\ V_3(U) & :-\ e_1(U,Z) \end{array}$$

To find a way of answering Q using the views, we first try to find a view that will *cover* the subgoal $e_1(X,Z)$ in the query. We realize that V_1 will suffice, so we create a *Minicon description* (MCD) for it. The MCD specifies that an atom $V_1(X,Y)$ will cover the subgoal $e_1(X,Z)$, but it also specifies that the atom will cover the first *two* subgoals in Q. Similarly, we create an MCD for V_2 and the third subgoal, and finally we combine the MCDs to produce the rewriting:

$Q'(X,Y) :-\ V_1(X,Y), V_2(X,Y)$

The important point to note is that the MCD may tell us that it covers more than the original subgoal for which it was created. Furthermore, MCDs will only be created when the views are guaranteed to be useful. For example, in the case of V_3, since the variable Z is projected from the answer, the view is useless and an MCD will not be created.

We now describe the construction of the rule-goal tree in detail, deferring a discussion of the order in which we expand nodes in the tree. Later, we describe several methods for optimizing the tree's construction.

4.2 Creating the rule-goal tree

The algorithm takes as input a conjunctive query $Q(\bar{X})$ that is posed at some peer, and a set of peer mappings and storage descriptions in $\mathcal{PPL}$. We first describe the algorithm for the case in which there are no comparison predicates in the PDMS or the query.

Step 1: The algorithm transforms every equality description into two inclusion mappings. It then transforms every inclusion description of the form $Q_1 \subseteq Q_2$ into the pair: $V \subseteq Q_2$, and $V \; :- \; Q_1$, where V is a new predicate name that appears nowhere else in the peer mappings.

Step 2: The algorithm builds a rule-goal tree T. When a node n in T is a goal node, it has a label $l(n)$ which is an atom whose arguments are variables or constants. The label $l(n)$ of a rule node is a peer description (except that the child of the root is labeled with the rule defining the query). Finally, a rule node n that is labeled with an inclusion description also has a label $unc(n)$: this label always includes at least the father of n, but may also include nodes that are siblings of its father goal node. As described earlier, the reason for this label is that an MCD can cover more that the subgoal for which it was created.

The root of T is labeled with the atom $Q(\bar{X})$, and it has a single rule-node child whose children are the subgoals of the query. The tree is constructed by iterating the following step, until no leaf nodes can be expanded further.

Choose an arbitrary leaf goal node n in T whose label is $l(n) = p(\bar{Y})$, and p is not a stored relation. Perform all the expansions possible in the following two cases. In either case, never expand a goal node n with a peer description that was used on the path from the root to n. This guarantees termination of the algorithm even in a cyclic PDMS.

1. Definitional expansion: this is the case where peer relations appear in GAV-style mappings. If p appears in the head of a definitional description r, expand n with the definition of p. Specifically, let r' be the result of unifying $p(\bar{Y})$ with the head of r. Create a child rule n_r, with $l(n_r) = r'$, and create one child goal-node for n_r for every subgoal of r' with the corresponding label. Existential variables in r' should be renamed so they are fresh variables that do not occur anywhere else in the tree constructed thus far.

2. Inclusion expansion: this is the case where peer relations appear in LAV-style mappings. If p appears in the right-hand side of an inclusion description or storage description r of the form $V \subseteq Q_1$ (or $V = Q_1$), we do the following. Let $n_1, \ldots, n_m$ be the children of the father node of n, and $p_1, \ldots, p_m$ be their corresponding

labels. Create an MCD for $p(\bar{Y})$ w.r.t. $p_1, \ldots, p_m$ and the description r. Recall that the MCD contains an atom of the form $V(\bar{Z})$ and the set of atoms in $p_1, \ldots, p_m$ that it covers.

Create a child rule node n_r for n labeled with r, and a child goal node n_g for n_r labeled with $V(\bar{Z})$. Set $unc(n_g)$ to be the set of subgoals covered by the MCD. Repeat this process for every MCD that can be created for $p(\bar{Y})$ w.r.t. $p_1, \ldots, p_m$ and the description r.

Step 3: we construct the solutions from T. The solution is a union of conjunctive queries over the stored relations. Each of these conjunctive queries represents one way of obtaining answers to the query from the relations stored at peers. Each of them may yield different answers unless we know that some sources are replicas of others.

Let us consider the simple case, where only definitional mappings are used, first. The answer would be the union of conjunctive queries, each with head $Q(\bar{X})$ and a body that can be constructed as follows. Let T' be a subset of T where we arbitrarily choose a *single* child at every goal node, and for which all leaves are labeled by stored relations. The body of a conjunctive query is the conjunction of all the leaves of T'.

To accommodate inclusion expansions as well, we create the conjunctive queries as follows. In creating T's we still choose a single child for every goal node. This time, we do not necessarily have to choose *all* the children of a rule node n. Instead, given a rule node n, we need to choose a subset of the children $n_1, \ldots, n_l$ of n, such that $unc(n_1) \cup \ldots \cup unc(n_l)$ includes all of the children of n.

Remark 1 We note that in some cases, an MCD may cover cousins or uncles of its father node, not only its own uncles. For brevity of exposition, we ignore this detail in our discussion. However, we note that we do not compromise completeness as a result. In the worst case, we obtain conjunctive rewritings that contain redundant atoms. □

Incorporating comparison predicates: as we stated earlier, comparison predicates provide a very useful mechanism for specifying constraints on domains of stored relations or peer relations, and therefore exploiting them can lead to significant pruning of the tree. When the query or the peer mappings and storage descriptions include comparison predicates we modify the algorithm as follows. We associate with each node n a constraint-label $c(n)$. The constraint label describes the conjunction of comparison predicates that are known to hold on the variables in $l(n)$.

As we build T, constraints get added and propagated to child nodes. Specifically, suppose we expand a node n with a definitional description r, and let $c_1 \wedge \ldots \wedge c_m$ be the comparison predicates in r. Then we set $c(r)$ to be $c(n) \wedge c_1 \wedge \ldots \wedge c_m$, and the labels of its children are the projections of $c(r)$ on the variables of the child.[3] When we expand a goal node with an inclusion peer description then the MCD will be created

[3] When a conjunction of constraints is projected on a subset of the variables, the result may be a disjunction of constraints. The algorithm can either choose to manipulate such disjunctions or approximate them by the least subsuming conjunctions.

w.r.t. the constraints in the parent and in the peer description. Finally, we do not expand a node in the tree if its label is not satisfiable (this implies that it can only yield the empty set of answers to Q).

In step 3, when we construct the conjunctive queries, we add to them the conjunction of their constraint labels. If the resulting conjunctive query in unsatisfiable, we discard it. Note that constraints can also be propagated *up* the tree (in the same spirit at the predicate move-around algorithm [LMS94]), thereby detecting additional unsatisfiable labels during the construction of the tree.

4.3 Optimizations

As explained earlier, a major challenge for reformulation in the context of PDMS is optimizing the construction of the rule-goal tree. Up to this point we described which nodes need to be in the tree. We now describe several optimization opportunities for this context. We note that some of these optimizations are borrowed from evaluation of datalog and logic programs, and we *lift* from the data level to the expression level to apply them during the construction of the tree.

Memoization: an obvious optimization is to memoize nodes as we construct the tree, and therefore not have to do any repeated work. If two nodes in the tree have isomorphic labels, then only one of them needs to be expanded. Note that this is especially important in the PDMS context, because we may have many paths between a pair of peers (or more precisely, between two relations located on different peers).

Detecting dead ends and useless paths: there are several cases in which we can prune certain parts of the rule-goal tree. The simplest one is when we detect that a peer relation R is not reachable from the stored relations, meaning that we cannot find data for R anywhere. One common case in which this may happen is if certain peers leave the system or are currently unavailable. In such a case, if the label of a goal node n is an atom of R, then we can mark its parent rule-node as unreachable as well. Furthermore, there is no need to expand any of the siblings of n, unless they are memoizing computation for other nodes in the tree.

Another way in which a node n can become unreachable is if its constraint label $c(n)$ is unsatisfiable. This may occur because the stored relations we have access to contain data that is known to be disjoint from what is requested in the query. The techniques of [HMSS01,SR92] can be used to efficiently detect unsatisfiable labels during the expansion of the tree.

Finally, a more subtle optimization can detect useless paths as follows. Suppose we have two sibling goal nodes with labels $p_1(\bar{X})$ and $p_2(\bar{Y})$, and suppose that p_1 appears in a *single* inclusion peer description of the form $V(\bar{Z}) \subseteq p_1(\bar{X}), p_2(\bar{Y})$, and that predicate p_2 appears on the right-hand side of numerous inclusion peer mappings. In this case, the only way to reformulate p_1 will be through V, and V already satisfies the subgoal $p_2(\bar{Y})$. Hence, there is no need to explore any of the other ways of reformulating p_2: they are all redundant.

Ordering the expansion of the tree: While the above three optimizations have significant potential, they must be complemented with a search strategy that orders the expansion of the nodes. The goal is to

find the dead ends as early as possible to maximize the pruning. Clearly, neither breadth-first or depth-first construction of the tree may yield the optimal construction order. It may be desirable, however, to defer (or complete avoid) following very long semantic paths. Since every reformulation step is likely to introduce some information loss, applying too many such steps will probably result in a poor refomulated query.

Finding first reformulations quickly: in many contexts, there will be a large number of reformulations, and hence an important optimization is to generate the *first* reformulations quickly so query execution can begin. Alternatively, we may assign a utility to reformulations, and try to generate the *good* ones first (as in [DH02] for the data integration context). We note that the execution engine or Piazza (based on the Tukwila query processor [IHW03]) answers queries as the data is streaming in from the network. Hence, coupled with finding the first optimizations fast, users experience little delay in obtaining first answers to the query.

Filter descriptions: when many data sources contain elements of the same type (e.g., publications, products), filter descriptions can be an important source of optimization. Filter descriptions are a mechanism that enables a peer to publish a *summary* of the values that it currently stores for one or more of its attributes. The summary can be used as a filter against data requests: if the data at the peer is disjoint with the query answers, the peer will not be used. Filter descriptions might take any of several forms. At one extreme, the summary can be the actual set of values, i.e., a projection from one or more tables. In other cases, it can be a histogram on the values. For example, in the case of publications, the summary can be the list of authors of the papers in the collection, or a histogram on the years of publication. (Note that when the set of available values can be described using a conjunction of arithmetic comparisons, these can be directly incorporated within peer mappings as comparison predicates.)

Filter descriptions can be used directly during tree construction in the same fashion as comparison predicates: each filter describes a set of possible values, and combining filters amounts to intersecting them. Alternatively, they can be used at a later stage, when obtaining rewritings or even during query optimization.

5 Experiments

This section describes an initial set of experiments concerning the performance of our reformulation algorithm. Our goals with the experiments were modest. First, our goal was to demonstrate that answering queries in a PDMS by following chains of peer descriptions is viable. Second, our goal is to identify the key bottlenecks in implementing a PDMS, in order to focus future work on optimization.

The major (yet, natural) impediment to performing experiments at this point is the lack of existing PDMS to test on. Hence, our experiments are based on a workload generator that produces PDMS for several reasonable topologies of PDMS. The generator takes as input two main parameters: (1) the number of peers $\mathcal{R}$ in the system, and (2) the expected diameter $\mathcal{L}$ of the PDMS (i.e., the longest chain of peer mappings that can be constructed). Intuitively, the diameter of the PDMS will correspond to the number

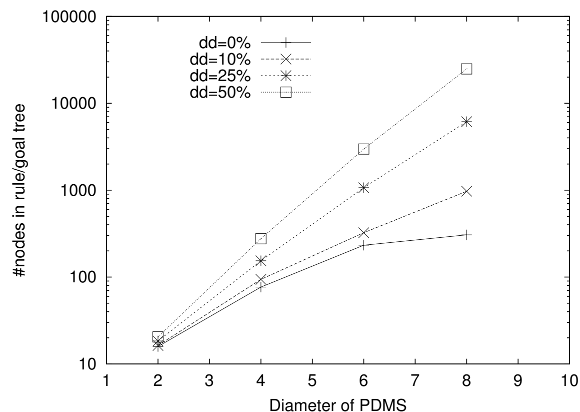

Fig. 3 The size of the rule/goal tree for different diameters of a 96-peer PDMS.

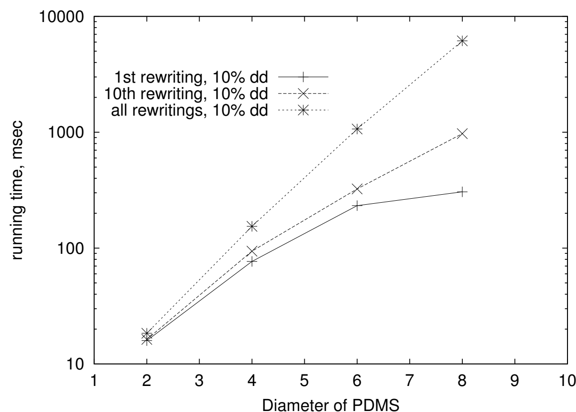

Fig. 4 The time to first answers (96 peers).

of levels of goal nodes in the tree. We call each such level a stratum, and to create the PDMS, we assign a number of peers to each stratum. The query is expressed in terms of the relations in the uppermost stratum, while the stored relations are in the lowest stratum.

The generator also controls the percent of peer mappings that are definitional versus inclusions. Finally, the right-hand sides of the peer mappings are usually chain queries over a set of relations that was selected randomly from the stratum below (for definitional mappings) and above (for inclusions). In our figures, each data point is generated from the average of 100 runs.

Because of the synthetic nature of the environment, our experiments cannot yield conclusive real-world results. However, they do clearly point out two important facts:

- The key bottleneck of the algorithm is the time to find the rewritings from the rule-goal tree (step 3), whereas step 2 scales up to rather large trees. Hence, an important issue is to tune the algorithm to produce the first rewritings as quickly as possible. We note that these findings are consistent with the experiments reported in [PH01], where the main factor affecting the running time of the MiniCon algorithm was the number of resulting rewritings.
- The main factor determining the size of the rule-goal tree is the diameter of the PDMS. In contrast, the number of peers at every stratum has a relatively little effect, because it is usually the case that most of them are irrelevant to a given query.

Figure 3 shows the size of the tree (number of nodes) as a function of the number of strata, and the percent of definitional peer mappings (in the figure, %dd denotes the percent of definitional mappings). As shown, with 8 strata, the size of the tree grows to 30,000 nodes. On average, the algorithm generates nodes at a rate of 1,000 per second (with relatively unoptimized code). We note that the size of the tree grows with the relative percent of definitional mappings. The reason for this is that we get more peer relations that are defined as unions of conjunctive queries, and hence a higher branching factor in the tree. In addition,

unfolding a definitional mapping replaces one peer relation with a set of relations (the body of the mapping) which also increases the branching factor.

Figure 4 shows that despite the large trees, the first rewritings can be found efficiently. For example, even with a diameter of 8, finding the first few rewritings can be done in under 3 seconds. Hence, we believe that in practice our algorithm will scale gracefully to large PDMS.

The experiments point to an important optimization problem for future work, namely, extracting the rewritings efficiently from the rule-goal tree. We have designed, but not implemented, a method that interleaves steps 2 and 3 of the algorithm. The method maintains a link structure among the nodes in the tree that records which combinations of nodes are used together in a conjunctive reformulation. Hence, the moment we construct a leaf, it is already linked to all relevant reformulations. A possible disadvantage of the method is higher memory requirements.

6 Global PDMS Considerations

Up to this point, we considered the problem of answering queries in a PDMS. However, the ad hoc nature of PDMSs raises several fundamental issues concerning the global management of semantic mappings. Since there is no central control of a PDMS, it can evolve substantially in structure and content, particularly as peers leave and join the system at will. We are interested in when a semantic mapping is redundant, and we would like to compose semantic mappings in order to speed up query reformulation. A fundamental problem in the global management of a PDMS is whether given two PDMSs, they are in some sense equivalent. In this section we first formalize this question, and then we provide two fundamental results regarding it. The first result shows that the answer subtly depends on the language we consider for queries, and the second shows that the problem is decidable in some important cases.

Formally, we consider two PDMSs, N_1 and N_2 and assume w.l.o.g. that they have the same set of peers, peer relations, and stored relations. We say that N_1 is *equivalent* to N_2 if for every instance of the stored relations, T, and every query Q posed at a peer, the set of certain answers of Q in N_1 is equal to the set of certain answers of Q in N_2. Depending on the query language for Q, we get different notions of equivalence. In our discussion we consider two cases: when Q ranges over all queries in First Order Logic (FO), and when Q ranges over conjunctive queries (CQ). To emphasize the language, we will refer to *FO-equivalence* and *CQ-equivalence*.

FO-equivalence differs from CQ-equivalence. To see this, assume two peers, the first holding a stored relation R and peer relation P_1, the second holding peer relations P_2, P_3. Define two PDMSs, N_1 and N_2:

$$N_1 : R(x) \subseteq P_1(x) \; P_1(x) \subseteq P_2(x) \; P_2(x) \subseteq P_3(x)$$
$$N_2 : R(x) \subseteq P_1(x) \; P_1(x) \subseteq P_2(x) \; P_1(x) \subseteq P_3(x)$$

The PDMS N_1 and N_2 are not FO-equivalent because the set of certain answers of the query $\{x \mid P_3(x) \wedge \forall z.(P_2(z) \Rightarrow P_3(z))\}$ in N_1 is all the tuples in R, while in N_2 is $\emptyset$. However, N_1 and N_2 are CQ-equivalent,

because any CQ query Q in P_1, P_2, P_3 is monotone in these three tables, hence, given an instance of the stored relation $I(R)$, the set of its certain answers is the result of Q on the smallest data instance compatible $I(R)$, if such a smallest instance exists. For both PDMS N_1 and N_2 the smallest instance exists and is given by $I(P_1) = I(P_2) = I(P_3) = I(R)$. Hence N_1 and N_2 are CQ-equivalent.

In practice, we are interested in a more general notion, which we call *relative equivalence*. Consider two PDMSs, N_1, N_2, and a set of peer-relation names, $\bar{P} = \{P_1, \ldots, P_m\}$. We say that N_1, N_2 are equivalent relative to $\bar{P}$ if, for every instance of the stored relations T and any query Q over the peer relations in $\bar{P}$, the set of certain answers of Q in N_1 is the same as that in N_2. For a set of peers $\bar{\mathcal{A}}$, we say that N_1 is equivalent to N_2 relative to $\bar{\mathcal{A}}$ if N_1 is equivalent to N_2 relative to all peer relation names occurring in $\bar{\mathcal{A}}$.

In the previous example, it can be seen that N_1 and N_2 are FO-equivalent relative to P_1, P_3: that is, if we can only ask queries over P_1 and P_3, then no FO query can distinguish N_1 from N_2.

Deciding relative equivalence is a key problem in PDMSs, because peers can join and leave at will, and we need to assess whether the system maintains the same semantics. Starting in some PDMS N_1, after some peers leave (or join) we end up in a PDMS N_2. The problem is whether N_1, N_2 are equivalent relative to $\bar{\mathcal{A}}$, where $\bar{\mathcal{A}}$ are the peers common to N_1 and N_2.

Our first result is providing an alternative characterization of the PDMS equivalence problem. While we defined PDMS equivalence in terms of answers to queries, the following theorem characterizes the problem in terms of the consistent data instances of the PDMS. A consequence of the theorem is to show that CQ-equivalence is different from FO-equivalence. The proof of the theorem is given in the appendix.

For a PDMS N and an instance of the stored relations T, we denote $Inst(N, T)$ the set of consistent data instances with N (Def. 1). If $\bar{P}$ is a subset of the peer relation names defined by N, and $I \in Inst(N, T)$, then $I(\bar{P})$ denotes the instances of the relations in $\bar{P}$.

Theorem 5 *Consider two PDMSs, N_1, N_2, and $\bar{P}$, a set of peer relation names.*

1. *N_1 and N_2 are FO-equivalent relative to $\bar{P}$ iff for every storage instance T, $\forall I \in Inst(N_1, T)$, $\exists J \in Inst(N_2, T)$ s.t. $I(\bar{P}) = J(\bar{P})$, and $\forall J \in Inst(N_2, T), \exists I \in Inst(N_1, T)$ s.t. $I(\bar{P}) = J(\bar{P})$.*
2. *N_1 and N_2 are CQ-equivalent relative to $\bar{P}$ only if for every storage instance T, $\forall I \in Inst(N_1, T)$, $\exists J \in Inst(N_2, T)$ s.t. $I(\bar{P}) \supseteq J(\bar{P})$, and $\forall J \in Inst(N_2, T), \exists I \in Inst(N_1, T)$ s.t. $I(\bar{P}) \subseteq J(\bar{P})$.*

Our main result of this section shows that checking PDMS equivalence is decidable in certain important cases. Importantly, the proof, given in the appendix, exploits the characterization of the PDMS equivalence given by Theorem 5.

Theorem 6 *Let N_1, N_2 be two PDMS with only inclusion peer descriptions that are also acyclic PDMS, and in which the storage descriptions do not have both projections and inclusions. Then:*

- *It is decidable whether N_1, N_2 are FO-equivalent.*

– *If the peer descriptions in both N_1, N_2 have only one atom on the left hand side, then for every set of peer relations $\bar{P}$, it is decidable whether N_1, N_2 are CQ-equivalent w.r.t. $\bar{P}$.*

7 Related Work

The idea of mediating between different databases using local semantic relationships is not new. Federated databases and cooperative databases have used the notion of inter-schema dependencies to define semantic relationships between databases in a federation (e.g., [LMR90,KLK91,RSK91,CL93]).

Most existing projects on semantic inter-operability such as SIMS [ACHK94], Information Manifold [LRO96], and InfoSleuth [Bea97] assume a single schema (or an ontology) that is used to pose queries. Data sources are described by their local schemas and capabilities. Given a query over the global schema, the system translates the query into the local schemas of the data sources. As a result, the user can access multiple heterogeneous data sources using a single schema.

The OBSERVER system [MKIA00] allows the user to select any of the known ontologies. The system can translate queries into other relevant ontologies automatically. Unlike Piazza, OBSERVER does not chain translation steps; rather every pair of ontologies are assumed to be mapped into each other. An IR-based approach to querying is supported. The system can return unsound as well as incomplete answers. OBSERVER tries to reduce the information loss caused by query translation and also provide a measure of information loss (both on recall and precision) with every query answer.

Similar to our work, Semantic Gossiping [ACMH02] considers a P2P environment where peers use different schemas, and define semantic mappings to other peers. The focus of their work, however, is on analyzing the resulting semantic network in order to design a single "consensus" schema and identify potential inconsistencies.

In [GHI+01] we described some of the challenges involved in building a PDMS, focusing on *intelligent data placement*, a technique for materializing views at nodes in the network in order to improve performance and availability. Maintenance of materialized views has recently received much attention in the context of data warehousing [AI95]. In [KNO+02] the authors study a variant of the data placement problem, and focus on intelligently caching and reusing queries in an OLAP environment. Recently, [BGK+02] described *local relational models* as a formalism for mediating between different peers in a PDMS, and a sound and complete algorithm for answering queries using the formalism, but do not describe the expressive power of the formalism compared to previous ones in the data integration literature.

The work [NOTZ03] describes PeerDB, a P2P-based system for distributed data sharing. Similar to Piazza, PeerDB does not require a global schema. Unlike Piazza, PeerDB does not use schema mappings for query reformulation. Instead, PeerDB employs an Information Retrieval -based approach for query reformulation. In their approach, a peer relation (and each of its columns) is associated with a set of keywords. Given a query over a peer schema, PeerDB reformulates the query into other peer schemas by matching

the keywords associated with the two schemas. Unlike Piazza, PeerDB does not need to chain multiple reformulation steps as the keywords in any pair of schemas can be matched directly. Also, in PeerDB, some reformulated queries may not be meaningful, and the user has to decide which queries are to be executed.

8 Conclusions

The concept of the peer data management system emphasizes not only an ad hoc, scalable, distributed peer-to-peer computing environment (which is compelling from a distributed systems perspective), but it provides an easily extensible, decentralized environment for sharing data with rich semantics. This is in contrast to data integration systems or other mediator architectures, which have a centralized mediated schema and administrator, and which impede small, point-to-point collaborations.

We presented a solution to schema mediation in peer data management systems. We described $\mathcal{PPL}$, a flexible mediation scheme for PDMSs, which uses previous mediation formalisms at the local level to form a network of semantically related peers. We characterized the theoretical limitations on answering queries in $\mathcal{PPL}$-PDMSs. Next, we described a query reformulation algorithm for $\mathcal{PPL}$. The primary contribution of the algorithm is that it combines both LAV- and GAV-style reformulation in a uniform fashion, and it is able to chain through multiple peer descriptions to reformulate a query. We described optimization methods for reformulation, and some initial experimental results that show its utility. The final result is a practical solution for schema mediation in PDMS. In addition, we introduced global considerations in a PDMS and addressed the PDMS-equivalence problem.

Future research includes extending our results to the XML data model and the XQuery query language (see [HITM03] for the first step in that direction). We are also looking at the problem of optimizing the topology of a PDMS to improve performance and decrease information loss. Some of the possible directions in this area are: identifying redundant mappings, minimizing the "diameter" of a PDMS, and detecting semantic paths that have a zero combined information flow. Addressing these problems requires the ability to compose mappings, which is a difficult problem on its own [HM03]. In addition, it is important to be able to identify ill-defined mappings that are either inconsistent or too "narrow". An automatic schema matching technique can be applied here to help the user define a better topology.

Acknowledgments

This work was funded in part by NSF ITR grants IIS-0205635 and IIS-9985114 and a gift from Microsoft.

References

[ACHK94] Y. Arens, C. Y. Chee, C.-N. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *IJCIS*, 2(2), 1994.

[ACMH02] K. Aberer, P. Cudre-Mauroux, and M. Hauswirth. A framework for semantic gossiping. *SIGMOD Record*, 31(4), 2002.

[ACPS96] S. Adali, K. Candan, Y. Papakonstantinou, and V.S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of SIGMOD*, pages 137–148, Montreal, Canada, 1996.

[AD98] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS*, pages 254–263, Seattle, WA, 1998.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Weseley, 1995.

[AI95] A.Gupta and I.S.Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Data Engineering Bulletin*, 18(2), 1995.

[BDSK89] Maurice Bruynooghe, Danny De-Schreye, and B. Krekels. Compiling control. *Journal of Logic Programming*, pages (6) 135–162, 1989.

[Bea97] R. Bayardo and et al. Infosleuth: Semantic integration of information in open and dynamic environments. In *SIGMOD*, Tucson, AZ, 1997.

[BGK+02] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing : A vision. In *Proceedings of the WebDB Workshop*, 2002.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[CL93] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, pages 55–62, 1993.

[DG97] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. of PODS*, pages 109–116, Tucson, Arizona., 1997.

[DH02] Anhai Doan and Alon Halevy. Efficiently ordering query plans for data integration. In *Proc. of ICDE*, 2002.

[FLM99] Marc Friedman, Alon Levy, and Todd Millstein. Navigational plans for data integration. In *Proceedings of AAAI*, 1999.

[FW97] M. Friedman and D. Weld. Efficient execution of information gathering plans. In *Proceedings of the International Joint Conference on Artificial Intelligence, Nagoya, Japan*, pages 785–791, 1997.

[GHI+01] Steven Gribble, Alon Halevy, Zachary Ives, Maya Rodrig, and Dan Suciu. What can databases do for peer-to-peer? In *ACM SIGMOD WebDB Workshop 2001*, 2001.

[GMPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems*, 8(2):117–132, March 1997.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4), 2001.

[HITM03] Alon Halevy, Zachary Ives, Igor Tatarinov, and Peter Mork. Piazza: Data management infrastructure for semantic web applications. In *Proc. of the Int. WWW Conf.*, 2003.

[HKWY97] Laura Haas, Donald Kossmann, Edward Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *Proc. of VLDB*, Athens, Greece, 1997.

[HM03] A. Halevy and J. Madhavan. Composing mappings among data sources. In *VLDB*, Berlin, Germany, 2003.

[HMSS01] Alon Y. Halevy, Inderpal Mumick, Yehoshua Sagiv, and Oded Shmueli. Static analysis in datalog extensions. *Journal of the ACM*, 48(5):971–1012, September 2001.

[IHW01] Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. Integrating network-bound XML data. *IEEE Data Engineering Bulletin Special Issue on XML*, 24(2), June 2001.

[IHW03] Zachary Ives, Alon Halevy, and Dan Weld. An xml query engine for network-bound data. *VLDB Journal, Special Issue on XML Query Processing*, 2003.

[KLK91] Ravi Krishnamurthy, Witold Litwin, and William Kent. Language features for interoperability of databases with schematic discrepancies. In *Proc. of SIGMOD*, pages 40–49, Denver, Colorado, 1991.

[KNO+02] P. Kalnis, W. Ng, B. Ooi, D. Papadias, and K. Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *Proc. of SIGMOD*, 2002.

[LKG99] Eric Lambrecht, Subbarao Kambhampati, and Senthil Gnanaprakasam. Optimizing recursive information gathering plans. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1204–1211, 1999.

[LMR90] Witold Litwin, Leo Mark, and Nick Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22 (3):267–293, 1990.

[LMS94] Alon Y. Levy, Inderpal Singh Mumick, and Yehoshua Sagiv. Query optimization by predicate movearound. In *Proc. of VLDB*, pages 96–107, Santiago, Chile, 1994.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, Bombay, India, 1996.

[MFK01] Ioana Manolescu, Daniela Florescu, and Donald Kossmann. Answering xml queries on heterogeneous data sources. In *Proc. of VLDB*, pages 241–250, 2001.

[MKIA00] E. Mena, V. Kashyap, A. Illarramendi, and A.Sheth. Retrieving and integrating data from multiple information sources. *IJCIS*, 9(4), 2000.

[Nap01] Napster. World-wide web: www.napster.com, 2001.
[NOTZ03] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. Peerdb: A p2p-based system for distributed data sharing. In *ICDE*, Bangalore, India, 2003.
[PH01] Rachel Pottinger and Alon Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB Journal*, 2001.
[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.
[RSK91] M. Rusinkiewicz, A. Sheth, and G. Karabatis. Specifying interdatabase dependencies ina multidatabase environment. *IEEE Computer*, 24:12, 1991.
[SBD+81] John M. Smith, Philip A. Bernstein, Umesh Dayal, Nate Goodman, T. Landers, K.W.T Lin, and E. Wong. Multibase – integrating heterogeneous distributed database systems. In *Proceedings of the National Computer Conference*, pages 487–499. AFIPS Press, Montvale, NJ, 1981.
[SR92] Divesh Srivastava and Raghu Ramakrishnan. Pushing constraint selections. In *Proc. of PODS*, pages 301–315, San Diego, CA., 1992.

Appendix: Proof of selected theorems

Proof of Theorem 1

We begin by sketching the proof of item (2). In the proof, we transform the peer and storage descriptions in N into inverse rules. More precisely, given a storage or peer description:

$$Q_1(\bar{X}) \subseteq Q_2(\bar{X})$$

we generate inverse rules of the form:

$$p_j(\bar{X}'_j) : -Q_1(\bar{X})$$

for every atom p_j in Q_2, as in the inverse rules described at the beginning of Sec. 2. To this, we add a rule for the query:

$$Answ(\bar{X}) : -Q(\bar{X})$$

This results in a datalog program, in which the stored relations are the EDBs and all peer relations, including $Answ$, are IDBs. The program is non-recursive (because N is acyclic) and has Skolem terms. Hence, it can be evaluated in polynomial time data complexity over the stored instance T. The certain answers are precisely the tuples in $Answ$ that do not contain any Skolem terms. □

Next, we prove item (1) by reduction from the implication problem for functional and inclusion dependencies.

The reduction results in a restricted version of our query answering problem which we call Unary Peer Answering Problem, UPA, having the following restrictions: (1) the PDMS, N, has only one stored relation, R, which is unary, (2) the stored instance, $T(R)$, has a single value a in R, (3) the query is also unary: $Q(y)$. We write (N, a, Q) for a UPA. It is easy to show that the set of certain answers is either $\emptyset$ or $\{a\}$: indeed, define I to be the instance in which every peer relation contains a unique tuple, $(a, a, \ldots, a)$. Q's answer on I is a, hence there are no other certain answers besides possibly a. (N, a, Q) is a decision problem, asking whether a is a certain answer. The following is easy to prove: (N, a, Q) is true iff (N, b, Q) is true, for any constants a and b, hence we will describe the UPA problem as (N, Q), without mentioning the constant a.

As background, given a relational schema $\bar{P}$, we consider *functional dependencies* (fd's) $P : \bar{C} \to D$ and *inclusion dependencies* (ind's) $P_1[\bar{C}] \subseteq P_2[\bar{D}]$ with the standard meaning, where $\bar{C}, \bar{D}$ are attribute names in the relations. For a database instance $I(\bar{P})$ we write $I(\bar{P}) \models \sigma$ if the fd or ind σ holds in $I(\bar{P})$; same for $I(\bar{P}) \models \Sigma$ for a set of fd's/ind's Σ. Given such a set Σ and a unary ind σ, we write $\Sigma \models \sigma$ if $\forall I(\bar{P})$, if $I(\bar{P}) \models \Sigma$ then $I(\bar{P}) \models \sigma$. It is known that it is undecidable whether $\Sigma \models \sigma$ (see [AHV95], Theorem 9.2.4, pp. 199). Thus, the *implication problem* for dependencies is undecidable.

Given an implication problem described by $\bar{P}$, Σ, σ we reduce it to the following UPA (N, Q). Assume σ to be $P_i[C] \subseteq P_j[D]$. The peer relations in N are $\bar{P}$, and there is a single stored relation R, and a single storage description in N:

$$R \subseteq P_i[C]$$

We now define the peer descriptions in N. There will be one peer description for every fd and every ind in Σ. Consider an fd $P : \bar{C} \to D$ first. The corresponding peer description is:

$$P[\bar{C}, D_1] \bowtie P[\bar{C}, D_2] \subseteq \sigma_{D_1=D_2}(P[\bar{C}, D_1] \bowtie P[\bar{C}, D_2]) \quad (1)$$

Consider an inclusion dependency $P_i[\bar{C}] \subseteq P_j[\bar{D}]$. The corresponding peer description is:

$$P[\bar{C}] \subseteq P[\bar{D}] \quad (2)$$

This completes the definition of N. The query Q is $P_j[D]$. Next we prove two lemmas.

Lemma 1 *If $\Sigma \models \sigma$ then the UPA (N, Q) is true.*

Proof Consider some constant a. Recall that in the UPA (N, Q) the stored instance is $T(R) = \{a\}$. Let I be a consistent data instance for the PDMS N. In particular I satisfies equations (1) and (2)). From the storage description it follows that a is in $P_i[C]$. From the peer descriptions it follows that $I(\bar{P}) \models \Sigma$. Hence, $I \models \sigma$. Hence a is in $P_j[D]$, i.e. in Q's answer. Since this was true for every consistent instance I, it follows that a is a certain answer.

Lemma 2 *If the UPA (N, Q) is true, then $\Sigma \models \sigma$.*

Proof Let $I'(\bar{P})$ be some instance satisfying Σ. It follows that $I'(\bar{P})$ also satisfies the peer descriptions (2) and (1). We want to prove that, for this instance, $P_i[C] \subseteq P_j[D]$. Let $a \in P_i[a]$. Since (N, a, Q) is true, it means that a is a certain answer of Q. Hence it is in $P_j[D]$.

These two lemmas complete the proof of item (1). □

Proof of Theorem 5

Proof (Sketch) Let $\bar{P} = \{P_1, \ldots, P_k\}$.

(1), "only if": Let Q be a query over the relations $\bar{P}$, let T be a stored instance, and let $\bar{a}$ be a certain answer for Q in N_1. We show that it is also a certain answer for Q in N_2. Let $J \in Inst(N_2, T)$, we need to show that $\bar{a} \in Q(J(\bar{P}))$. This follows from the fact that $\exists I \in Inst(N_1, T)$ s.t. $I(\bar{P}) = J(\bar{P})$ and $\bar{a} \in Q(I(\bar{P}))$.

"if": Let T be a storage instance and $I \in Inst(N_1, T)$. We will write a boolean query $Q(\bar{P})$ which is true only on instances that are different (non-isomorophic) to $I(\bar{P})$. Consider all data values that occur in T and $I(\bar{P})$: $a_1, \ldots, a_n$, and assume that the first n occur in T and the remainig $m - n$ do not occur in T. Then Q is an n-ary query, $Q(x_1, \ldots, x_n) = \neg Q_I$, where $Q_I = (\exists x_{m+1} \ldots \exists x_n.(\bigwedge_{i<j}(x_i \neq x_j)) \wedge C_1 \wedge \ldots C_k)$, where each C_i checks precisely the tuples in $I(P_i)$. For example, assuming $I(P_i) = \{(a_2, a_3), (a_5, a_5)\}$, then $C_i = P_i(x_2, x_3) \wedge P_i(x_5, x_5) \wedge (\forall x, y.(P(x, y) \Rightarrow (x = x_2 \wedge y = x_3) \vee (x = x_5 \wedge y = x_5)))$. The set of certain answers of Q in N_1 is the empty set, since $Q(I(\bar{P})) = \emptyset$. Hence it must be the same in N_2. Hence, $\exists J \in Inst(N_2, T)$ s.t. $(a_1, \ldots, a_n) \notin Q(J)$, i.e. J satisfies $Q_I(a_1, \ldots, a_n)$. This means that $J(\bar{P})$ is ismomorphic to $I(\bar{P})$, with the isomorphism preserving the values in T, hence it can be replaced by some J' s.t. $I(\bar{P}) = J(\bar{P})$ (details omitted).

(2), "only if": this is shown as in (1), but we use the monotonicity of Q. We leave the details to the full paper.

Proof of Theorem 6

Proof (Sketch) The algorithms for deciding FO-equivalence consists of enumerating all storage descriptions T_0 and instances I_0 up to a certain size, and checking whether $I_0 \in Inst(N_1, T_0) \iff I_0 \in Inst(N_2, T_0)$. We need to prove that we can find a bound for the size of T_0 and I_0. For that suppose that T, I are such that $I \in Inst(N_1, T)$ and $I \notin Inst(N_2, T)$. We will construct a "small" subset $T_0 \subseteq T, I_0 \subseteq I$ with the same property. Since $I \notin Inst(N_2, T)$, there exists a peer description $Q_1 \subseteq Q_2$ which is violated by I. More precisely there exists an answer $\bar{a} \in Q_1(I)$ s.t. $\bar{a} \notin Q_2(I)$. Since Q_1 is a conjunctive query, there is a "small" subset I' of I s.t. $\bar{a} \in Q_1(I')$: when constructing $I_0 \subseteq I$ we will ensure that $I' \subseteq I_0$: this implies both $\bar{a} \in Q_1(I_0)$ and $\bar{a} \notin Q_2(I_0)$, hence it is the case that $I_0 \notin Inst(N_2, T_0)$. Now we turn to N_1, and construct a "small" stored relation T_0 and instance I_0 that is still consistent with T_0 and contains I'. For that we partition the relation names into strata, s.t. for every k, if $Q_1 \subseteq Q_2$ is a peer or storage description at N_1, and all relations in Q_1 are in strata k and lower, then all relations in Q_2 are in strata $k+1$ or higher. It follows that the stored relations are in stratum 0, all others in higher strata. Now we define I_0 for each stratum. At stratum 0 we define $I_0(R) = I'(R)$ for each stored relation R: this defines T_0. At stratum $k+1$, we consider all peer or storage descriptions $Q_1 \subseteq Q_2$ defining this stratum. Let $\bar{a}_1, \ldots, \bar{a}_n$ be all the answer in $Q_1(I_0)$. For each i, $\bar{a}_i \in Q_2(I)$. Hence there exists a "small" subset of I on the relations in stratum $k+1$ on which Q_2 still returns $\bar{a}_i$. Take I_0 at this stratum to be the union of all these subsets and of I'. Thus, we

construct I_0 stratum by stratum, ensure at each step that the peer descriptions in N_1 at that stratum are satisfied. This completes the proof. Notice that at each new stratum we increase the size of I_0 by a factor equal to the size of the queries in N_1, hence I_0 will have a size which is exponential in N_1.

The algorithm for deciding CQ-equivalence consists of eliminating all interemediate levels and "flattening" both PDMS N_1, N_2, then checking equivalence of flat PDMS. Flattening for N_1 proceeds as follows (for N_2 is similar, of course). Consider some peer relation name P. Assume it occurs in the peer description $P \subseteq Q_2$ and in several peer descrptions of the form $Q_1 \subseteq Q(P)$. Define N_1' to be obtained from N_1 by deleting the peer description $P \subseteq Q_2$ and replacing each $Q_1 \subseteq Q(P)$ with $Q_1 \subseteq Q(P \cap Q_2)$, when P is in the set $\bar{P}$ relative to which we check equivalence, or with $Q_1 \subseteq Q(Q_2)$ if $P \notin \bar{P}$. Here we assumed Q to be expressed as a select-project-join query, hence $Q(P \cap Q_2)$ means "substitute the base relation P with $P \cap Q_2$". It is easy to prove, using Theorem 5 that N_1 and N_1' are CQ-equivalent. Indeed, let J be a consistent instance for N_1'. We construct out of it a smaller consistent instance I for N_1 by defining I to be identical to J except for the relation P, where $I(P) = J(P) \cap Q_2(J)$, in the case $P \in \bar{P}$, or $I(P) = Q_2(J)$ otherwise. The other direction is trivial: given I, take J to be identical to I.

Example 6 For a simple example, consider the PDMS N_1 in presented earlier, and suppose we want to eliminate P_2. The construction above yields:

$$N_1' : R(x) \subseteq P_1(x) \; P_1(x) \subseteq P_2(x), P_3(x)$$

which is semantically equivalent to N_2. For a more complex example, consider the PDMS:

$$\begin{aligned} N_1 : A(x,y) &\subseteq B(x,z), B(z,y) \\ B(x,y) &\subseteq C(x,z), C(z,y) \end{aligned}$$

and assume we eliminate B. We get:

$$\begin{aligned} N_1 : A(x,y) \subseteq B(x,z), &C(x,u), C(u,z), \\ &B(z,y), C(y,v), C(v,y) \end{aligned}$$

Returning to the proof of Th. 6, after repeatedly applying the elimination procedure to N_1 we arrive at a "flat" PDMS which has a single stratum, that is, it consists only of storage descriptions. Similarly, N_2 will be transformed into something similar, i.e.:

$$N_1 : \; R_1 \subseteq Q_1(\bar{P}), \ldots, R_n \subseteq Q_n(\bar{P}) \tag{3}$$

$$N_2 : \; R_1 \subseteq Q_1'(\bar{P}), \ldots, R_n \subseteq Q_n'(\bar{P}) \tag{4}$$

Notice that only the relations $\bar{P}$ occur in the right-hand side, relative to which we check containment, since all the others have been eliminated. The following proposition concludes the proof of the theorem.

Proposition 1 *Consider two flat PDMS N_1, N_2 consisting only of storage descrptions (3), (4). Then N_1 is CQ-equivalent to N_2 iff N_1 is FO-equivalent to N_2 iff for each i, Q_i is equivalent (as a conjunctive query) to Q'_i.*

Proof Clearly if $Q_1 \equiv Q'_1, \ldots, Q_n \equiv Q'_n$ then N_1, N_2 are both FO- and CQ-equivalent. For the converse, assume they are CQ equivalent. Rename all variables in all $2n$ queries to be distinct. For each $i = 1, \ldots, n$, define Q to be the conjunctive query Q_i, and consider the stored instance T for which R_i contains precisely one tuple given by the head variables, $\bar{x}_i$, in $R_i(\bar{x}_i) \subseteq Q_i(\bar{x}_i)$, and all the other R_j's are empty. Clearly $\bar{x}_i$ is a certain answer for Q in N_1, hence it is a certain answer for Q in N_2. Consider the instance I defined by the body of Q'_i: this is a consistant instance for N_2 w.r.t. T, hence $\bar{x}_i \in Q(I)$. This gives us a containment mapping from $Q_i(= Q)$ to $Q'(= I)$, proving $Q'_i \subseteq Q_i$. The other direction is proved similarly.