Query Answering in Peer-to-Peer Data Exchange Systems

Leopoldo Bertossi and Loreto Bravo

Carleton University, School of Computer Science, Ottawa, Canada
{bertossi,lbravo}@scs.carleton.ca

Abstract. The problem of answering queries posed to a peer who is a member of a peer-to-peer data exchange system is studied. The answers have to be consistent wrt to both the local semantic constraints and the data exchange constraints with other peers; and must also respect certain trust relationships between peers. A semantics for *peer consistent answers* under exchange constraints and trust relationships is introduced and some techniques for obtaining those answers are presented.

1 Introduction

In this paper the problem of answering queries posed to a peer who is a member of a peer-to-peer data exchange system is investigated. When a peer `P` receives a query and is going to answer it, it may need to consider both its own data and the data stored at other peers' sites if those other peers are related to `P` by data exchange constraints (DECs). Keeping the exchange constraints satisfied, may imply for peer `P` to get data from other peers to complement its own data, but also not to use part of its own data. In which direction `P` goes depends not only on the exchange constraints, but also on the *trust relationships* that `P` has with other peers. For example, if `P` trust another peer `Q`'s data more than its own, `P` will accommodate its data to `Q`'s data in order to keep the exchange constraints satisfied. Another element to take into account in this process is a possible set of local semantic constraints that each individual peer may have.

Given a network of peers, each with its own data, and a particular peer `P` in it, a *solution for* `P` is -loosely speaking- a global database instance that respects the exchange constraints and trust relationships `P` has with its immediate neighbors and stays as close as possible to the available data in the system. Since the answers from `P` have to be consistent wrt to both the local semantic constraints and the data exchange constraints with other peers, the *peer consistent answers* (PCAs) from `P` are defined as those answers that can be retrieved from `P`'s portion of data in *every* possible solution for `P`. This definition may suggest that `P` may change other peers' data, specially of those he considers less reliable, but this is not the case. The notion of solution is used as an auxiliary notion to characterize the correct answers from `P`'s point of view. Ideally, `P` should be able to obtain its peer consistent answers just by querying the already available local instances. This resembles the approach to *consistent query answering* (CQA) in databases [1,4], where answers to queries that are consistent with given ICs are computed without changing the original database.

We give a precise semantics for peer consistent answers to first-order queries. First for the *direct case*, where transitive relationships between peers via ECs are not automatically considered; and at the end, the *transitive case*. We also illustrate, by means of extended and representative examples, mechanisms for obtaining PCAs. One of them is first order (FO) query rewriting, where the original query is transformed into a new query, whose standard answers are the PCAs to the original one. This methodology has intrinsic limitations. The second, more general, approach is based on a specification of the solutions for a peer as the stable models of a logic program, which captures the different ways the system stabilizes after satisfying the DECs and the trust relationships.

An instance r of a relational database can be seen a a set of ground atoms. Accordingly, an instance r' is a *repair* of instance r wrt a set of integrity constraints IC if $r' \models IC$ and r' minimally differs from r in terms of inclusion of set of tuples [1].

2 A Framework for P2P Data Exchange

In this section we will describe the framework we will use to formalize and address the problem of query answering in P2P systems.

Definition 1. A *P2P data exchange system* $\mathfrak{P}$ consists of:

(a) A finite set $\mathcal{P}$ of peers, denoted by $\mathtt{A}, \mathtt{B}, \mathtt{C}, ..., \mathtt{P}, \mathtt{Q}, ...$
(b) For each peer $\mathtt{P}$, a database schema $\mathcal{R}(\mathtt{P})$, that includes a domain $D(\mathtt{P})$, and relations $R(\mathtt{P}), ...$. However, it may be convenient to assume that all peers share a common, fixed, possibly infinite domain, D. Each $\mathcal{R}(\mathtt{P})$ determines a FO language $\mathcal{L}(\mathtt{P})$. We assume that the schemas $\mathcal{R}(\mathtt{P})$ are disjoint, being the domains the only possible exception. $\mathcal{R}$ denotes the union of the $\mathcal{R}(\mathtt{P})$s.
(c) For each peer $\mathtt{P}$, a database instance $r(\mathtt{P})$ corresponding to schema $\mathcal{R}(\mathtt{P})$.
(d) For each peer $\mathtt{P}$, a set of $\mathcal{L}(\mathtt{P})$-sentences $IC(\mathtt{P})$ of ICs on $\mathcal{R}(\mathtt{P})$.
(e) For each peer $\mathtt{P}$, a collection $\Sigma(\mathtt{P})$ of *data exchange constraints* $\Sigma(\mathtt{P}, \mathtt{Q})$ consisting of sentences written in the FO language for the signature $\mathcal{R}(\mathtt{P}) \cup \mathcal{R}(\mathtt{Q})$, and the $\mathtt{Q}$'s are (some of the) other peers in $\mathcal{P}$.
(f) A relation $trust \subseteq \mathcal{P} \times \{less, same\} \times \mathcal{P}$, with the intended semantics that when $(\mathtt{A}, less, \mathtt{B}) \in trust$, peer $\mathtt{A}$ trusts itself less than $\mathtt{B}$; while $(\mathtt{A}, same, \mathtt{B}) \in trust$ indicates that $\mathtt{A}$ trusts itself the same as $\mathtt{B}$. In this relation, the second argument functionally depends on the other two. By default a peer trusts its own data more than that of other peers. □

Each peer $\mathtt{P}$ is responsible for maintaining its material instance wrt $IC(\mathtt{P})$, independently from other peers. In particular, we assume $r(\mathtt{P}) \models IC(\mathtt{P})$. However, when local data is virtually changed to accommodate to other peers' data, the local ICs could be virtually violated. It is possible to keep the local ICs satisfied also at query time by using methodologies for consistent query answering, i.e. for consistently answering queries in databases that fail to satisfy certain ICs [4]. A peers may submit queries to other peers in accordance with the restrictions imposed its DECs and using the other peer's relations appearing in them.

Definition 2. (a) We denote with $\overline{\mathcal{R}}(\mathtt{P})$ the schema consisting of $\mathcal{R}(\mathtt{P})$ extended with the other peers' schemas that contain predicates appearing in $\Sigma(\mathtt{P})$. (b) For a peer $\mathtt{P}$

and an instance r on $\mathcal{R}(\texttt{P})$, we denote by $\bar{r}$, the database instance on $\overline{\mathcal{R}}(\texttt{P})$, consisting of the union of r with all the peers' instances whose schemas appear in $\overline{\mathcal{R}}(\texttt{P})$. (c) If r is an instance over a certain schema $\mathcal{S}$ and $\mathcal{S}'$ is a subschema of $\mathcal{S}$, then $r|\mathcal{S}'$ denotes the restriction of r to $\mathcal{S}'$. In particular, if $\mathcal{R}(\texttt{P}) \subseteq \mathcal{S}$, then $r|\texttt{P}$ denotes the restriction of r to $\mathcal{R}(\texttt{P})$. (d) We denote by $\mathcal{R}(\texttt{P})^{less}$ the union of all schemas $\mathcal{R}(\texttt{Q})$, with $(\texttt{P}, \mathit{less}, \texttt{Q}) \in \mathit{trust}$. Analogously is $\mathcal{R}(\texttt{P})^{same}$ defined. □

From the perspective of a peer P, its own database may be inconsistent wrt the data owned by another peer Q and the DECs in $\Sigma(\texttt{P},\texttt{Q})$. Only when P trust Q the same as or more than itself, it has to consider Q's data. When P queries its database, these inconsistencies may have to be taken into account. Ideally, the answers to the query obtained from P should be consistent with $\Sigma(\texttt{P},\texttt{Q})$ (and its own ICs $\Sigma(\texttt{P})$). In principle, P, who is not allowed to change other peers' data, could try to repair its database in order to satisfy $\Sigma(\texttt{P}) \cup \mathit{IC}(\texttt{P})$. This is not a realistic approach. Rather P should solve its semantic conflicts or incompleteness of data at query time, when it queries its own database and those of other peers. Any answer obtained in this way should be sanctioned as correct wrt to a precise semantics.

The semantics of peer consistent query answers for a peer P is given in terms of all possible minimal, virtual, simultaneous repairs of the local databases that lead to a satisfaction of the DECs while respecting P's trust relationships to other peers. This repair process may lead to alternative global databases called the *solutions* for P. Next, the peer consistent answers from P are those that are invariant wrt to all its solutions. A peer's solution captures the idea that only some peers' databases are relevant to P, those whose relations appear in its trusted exchange constraints, and are trusted by P at least as much as it trusts its own data. In this sense, this is a "local notion", because it does not take into consideration transitive dependencies (but see Section 5).

Definition 3. (direct case) Given a peer P in a P2P data exchange system and an instance r on $\mathcal{R}$, an instance r' on $\mathcal{R}$ is a *solution for* P if r' is a repair of r wrt to $\Sigma(\texttt{P}) \cup \mathit{IC}(\texttt{P})$ that does not change the more trusted relations, more precisely: (a) $r' \models \bigcup\{\Sigma(\texttt{P},\texttt{Q}) \mid (\texttt{P}, \mathit{less}, \texttt{Q}) \text{ or } (\texttt{P}, \mathit{same}, \texttt{Q}) \in \mathit{trust}\} \cup \mathit{IC}(\texttt{P})$; (b) $r'|P = r|P$ for every predicate $P \in \mathcal{R}(\texttt{Q})$, where Q is a peer with $(\texttt{P}, \mathit{less}, \texttt{Q}) \in \mathit{trust}$; (c) r' minimally differs from r in the sense that $(r' \smallsetminus r) \cup (r \smallsetminus r')$ is minimal under set inclusion among those instances that satisfy (a) and (b). □

Intuitively, a solution for P repairs the global instance wrt the DECs with peers that P trusts more than or the same as itself, but leaving unchanged the tables that belong to more trusted peers. As a consequence of the definition, tables belonging to peers that are not related to P or are less trustable are not changed. That is, P tries to change its own tables according to what the dependencies to more or equally trusted peers prescribe.

The solutions for a peer are used as a conceptual, auxiliary tool to characterize the peer consistent answers; and we are not interested in them *per se*. Solutions are virtual and may be only partially computed if necessary, if this helps us to compute the correct answers obtained in/from a peer. The "changes" that are implicit in the definition of solution via the set differences are expected to be minimal wrt to sets of tuples which are inserted/deleted into/from the tables.

In these definitions we find clear similarities with the characterization of consistent query answers in single relational databases [4]. However, in P2P query answering, repairs may involve data associated to different peers, and also a notion of priority that is related to the trust relation.

Example 1. Consider a P2P data exchange system with peers P1, P2, P3, and schemas $\mathcal{R}_i = \{R^i, \ldots\}$. (a) Instances $r(\texttt{P1}) = \{R^1(a,b), R^1(s,t)\}$, $r(\texttt{P2}) = \{R^2(c,d), R^2(a,e)\}$, $r(\texttt{P3}) = \{R^3(a,f), R^3(s,u)\}$.
(b) $\mathit{trust} = \{\ (\texttt{P1}, \mathit{less}, \texttt{P2}),\ (\texttt{P1}, \mathit{same}, \texttt{P3})\ \}$. (c) $\Sigma(\texttt{P1},\texttt{P2}) = \{\ \forall xy(R^2(x,y) \rightarrow R^1(x,y))\ \}$; $\Sigma(\texttt{P1},\texttt{P3}) = \{\ \forall xyz(R^1(x,y) \wedge R^3(x,z) \ \rightarrow\ y = z)\ \}$. Here, the global instance is $r = \{R^1(a,b), R^1(s,t), R^2(c,d), R^2(a,e), R^3(a,f), R^3(s,u)\}$. It has two repairs according to Definition 3, namely $r' = \{R^1(a,b), R^1(s,t), R^1(c,d), R^1(a,e), R^2(c,d), R^2(a,e)\ \}$;
and $r'' = \{\ R^1(a,b), R^1(c,d), R^1(a,e), R^2(c,d), R^2(a,e), R^3(s,u)\}$. □

Definition 4. Given a FO query $Q(\bar{x}) \in \mathcal{L}(\texttt{P})$ posed to P, a ground tuple $\bar{t}$ is a *peer consistent* answer to Q for P iff $r'|\texttt{P} \models Q(\bar{t})$ for every solution r' for P. □

Example 2. (example 1 continued) The query $Q: R^1(x,y)$ posed to P1 has as peer consistent answers the tuples: (a,b), (c,d), (a,e), because those are the tuples found in relation R^1 in the restriction to P1's schema in every solution for P. □

Notice that this definition is relative to a fixed peer, and not only because the query is posed to one peer and in its query language, but also because this notion is based on the "direct or local" notion of solution for a single peer, which considers its "direct neighbors" only. This is a first step towards the general case of transitive dependencies, that will be explored in Section 5. However, this restricted case is the basis for the transitive case, because P does not see beyond its neighbors; and when P requests data to a neighbor, say Q, the latter may have to find local solutions of its own by considering its direct neighbors. The transitive case has to combine these local solutions.

Peer consistent answers to queries can be obtained by using techniques similar to those for CQA, e.g. query rewriting [1, 4]. However, there are important differences, because now we have some fixed predicates in the repair process.

Example 3. (example 1 continued) If P1 is posed the query $Q: R^1(x,y)$, asking for the tuples in relation R^1, its answers can be obtained through the rewritten query Q': $[R^1(x,y) \wedge \forall z_1((R^3(x,z_1) \wedge \neg\exists z_2 R^2(x,z_2)) \rightarrow z_1 = y)] \vee R^2(x,y)$, which requires from P1 to submit queries to its peers. The final answers are (a,b), (c,d), (a,e), precisely the answers obtained in Example 2. □

Notice that a query Q may have peer consistent answers for a peer which are not answers to Q when the peer is considered in isolation, which makes sense, because the peer may import data from other peers.[1]

This query rewriting approach differs from the one used for CQA. In the latter case, literals in a query are resolved (by *resolution*) against ICs in order to generate residues

[1] Another difference with CQA, where all consistent answers are answers to the original query; at least for conjunctive queries and *generic* ICs [4].

that are iteratively appended as extra conditions to the query. In the case of P2P data systems, the query may have to be modified in order to include new data that is located at a different peer's site. This cannot be achieved by imposing extra conditions alone, but instead, by relaxing the query in some sense. Since query answering in P2P systems includes sufficiently complex cases of CQA, a FO query rewriting approach to P2P query answering is bound to have limitations in terms of completeness [4]. Instead, we will now propose a more general methodology based on answer set programming.

3 Referential Data Exchange Constraints

An answer set programming approach to the specification of solutions for a peer can be developed. Those specifications will be similar to those of repairs of single relational databases under referential integrity constraints [2]. However, as we have seen, there are important differences with CQA.

In most applications we may expect the DECs for a peer to be inclusion dependencies or referential constraints, which will be used by this peer to either import data from or to validate its own data with another peer. We now give an example of an even more more involved referential constraint that shows the main issues around this kind of specifications.

Example 4. Consider a P2P system with peers $\mathtt{P}$ and $\mathtt{Q}$, with schemas $\{R_1(\cdot,\cdot), R_2(\cdot,\cdot)\}$, $\{S_1(\cdot,\cdot), S_2(\cdot,\cdot)\}$, resp.; and assume that $\mathtt{P}$ is querying its database subject to its DEC that mixes tables of the two peers on each side of the implication:

$$\forall x \forall y \forall z \exists w (R_1(x,y) \wedge S_1(z,y) \ \rightarrow \ R_2(x,w) \wedge S_2(z,w)), \tag{1}$$

We consider the case where $(\mathtt{P}, \mathit{less}, \mathtt{Q}) \in \mathit{trust}$, i.e. $\mathtt{P}$ considers $\mathtt{Q}$'s data more reliable than its own. If (1) is satisfied by the combination of the data in $\mathtt{P}$ and $\mathtt{Q}$, then the current global instance constitutes $\mathtt{P}$'s solution. Otherwise, alternative solutions for $\mathtt{P}$ have to be found, keeping $\mathtt{Q}$'s data fixed in the process. This is the case, when there are ground tuples $R_1(d,m) \in r(\mathtt{P}), S_1(a,m) \in r(\mathtt{Q})$, such that for no t it holds both $R_2(d,t) \in r(\mathtt{P})$ and $S_2(a,t) \in r(\mathtt{Q})$.

Obtaining peer consistent answers for peer $\mathtt{P}$ amounts to virtually restoring the satisfaction of (1), by virtually modifying $\mathtt{P}$'s data. In order to specify $\mathtt{P}$'s modified relations, we introduce virtual versions R_1', R_2' of R_1, R_2, containing the data in peer $\mathtt{P}$'s solutions. In consequence, at the solution level, we have the relations R_1', R_2', S_1, S_2. Since $\mathtt{P}$ is querying its database, its original queries will be expressed in terms of relations R_1', R_2' only (plus, possibly, built-ins).

The contents of the virtual relations R_1', R_2' are obtained from the material sources R_1, R_2, S_1, S_2.[2] Since S_1, S_2 are fixed, the satisfaction of (1) requires R_1' to be a subset of R_1, and R_2', a superset of R_2. The specification of these relations is done in extended disjunctive logic programs with answer set (stable model) semantics [13]. The first rules for the specification program Π are:

$$R_1'(x,y) \leftarrow R_1(x,y),\ \mathit{not}\ \neg R_1'(x,y). \quad R_2'(x,y) \leftarrow R_2(x,y),\ \mathit{not}\ \neg R_2'(x,y), \tag{2}$$

[2] We can observe that the virtual relations can be seen as virtual global relations in a virtual data integration system [18, 20].

which specify that, by default, the tuples in the source relations are copied into the new virtual versions, but with the exception of those that may have to be removed in order to satisfy (1) (with R_1, R_2 replaced by R_1', R_2'). Some of the exceptions for R_1' are specified by

$$\neg R_1'(x,y) \leftarrow R_1(x,y), S_1(z,y),\ \mathit{not}\ aux_1(x,z),\ \mathit{not}\ aux_2(z). \tag{3}$$

$$aux_1(x,z) \leftarrow R_2(x,w), S_2(z,w). \qquad aux_2(z) \leftarrow S_2(z,w). \tag{4}$$

That is, $R_1(x,y)$ is deleted if it participates in a violation of (1) (what is captured by the first three literals in the body of (3) plus the first rule in (4)), and there is no way to restore consistency by inserting a tuple into R_2, because there is no possible matching tuple in S_2 for the possibly new tuple in R_2 (what is captured by the last literal in the body of (3) plus the second rule in (4)). In case there is such a tuple in S_2, we either delete a tuple from R_1 or insert a tuple into R_2:

$$\neg R_1'(x,y) \vee R_2'(x,w) \leftarrow R_1(x,y), S_1(z,y),\ \mathit{not}\ aux_1(x,z), S_2(z,w), \\ \mathit{choice}((x,z),w). \tag{5}$$

That is, in case of a violation of (1), when there is tuple of the form (a,t) in S_2 for the combination of values (d,a), then the *choice operator* [14] non deterministically chooses a unique value for t, so that the tuple (d,t) is inserted into R_2 as an alternative to deleting (d,m) from R_1. The *choice* predicate can be replaced by a standard predicate plus extra rules that choose a unique value for t [14]. No exceptions are specified for R_2', which makes sense since R_2' is a superset of R_2. Then, the negative literal in the body of (2) can be eliminated. However, new tuples can be inserted into R_2', what is captured by rule (5). Finally, the program must contain as facts the tuples in the original relations R_1, R_2, S_1, S_2.

If P equally trusts itself and Q, both P and Qs' relations are flexible when searching for a solution. The program becomes more involved, because now S_1, S_2 may also change; and virtual versions for them must be specified. □

This example shows the main issues in the specification of a peer's solutions under referential DECs. The program with choice operator can be translated into one with standard answer set (or stable model) semantics [14]; and the solutions are in one to one correspondence with the answer sets of the program. Actually, each answer set S corresponds to a solution $r'(S)$ for peer P which coincides with the original, material, global instance on the tables other than R_1, R_2, whereas for the latter the contents are of the form $\{\bar{t} \mid R_i'(\bar{t}) \in S\}, i = 1, 2$, resp. The absence of solutions for a peer is captured through the non existence of answer sets for program Π.

Since program Π represents in a compact form all the solutions for a peer, the peer consistent answers from a peer can be obtained by running a query program expressed in terms of the virtually repaired tables, in combination with the specification program Π. For this the combined program is run under the skeptical answer set semantics, for which a system like DLV [19] can be used. For example, the query $\mathcal{Q}(x,z)\ :\ \exists y(R_1(x,y) \wedge R_2(z,y))$ issued to peer P, would be peer consistently answered by running the query program $Ans_\mathcal{Q}(x,z) \leftarrow R_1'(x,y), R_2'(x,y)$ together with program Π. Although only (the new versions of) P's relations appear in the query, the program may make P import Q's data.

In the presence of referential DECs, the *choice operator* may have to choose values from the infinite underlying domain, but outside the active domains. There are several options, some of them already considered for CQA: (a) Live with an open infinite domain; (b) Assign null values without propagation through DECs [2]; (c) Consider an appropriate finite and closed proper superset of the active domains [6]; (d) Introduce fresh constants whenever needed from a separate domain [8]. We do not commit to any of these options here, but this choice and the class of referential ECs (e.g. presence cycles) may determine, e.g. decidability of peer consistent answering [7–9, 16].

If a peer $\mathtt{P}$ has local ICs $IC(\mathtt{P})$ to be satisfied, also at query time, then the program that specifies its solutions should take care of its ICs. A simple but radical way of doing this consists in using program denial constraints. If in Section 3 we had for peer $\mathtt{P}$ the local functional dependency (FD) $\forall x \forall y \forall z(R_1(x,y) \wedge R_1(x,z) \rightarrow y = z)$, then program would include the program constraint $\leftarrow R_1(x,y), R_1(x,z), y \neq z$, having the effect of pruning those solutions that do not satisfy the FD. However, a more flexible -or "robust" [11]- alternative for keeping the local ICs satisfied, consists in having the specification program split in two layers, where the first one builds the solutions, without considering the local ICs, and the second one, repairs the solutions wrt the local ICs, as done with single inconsistent relational databases [2]. A more uniform approach consists in identifying $IC(\mathtt{P})$ with $\Sigma(\mathtt{P},\mathtt{P})$ and considering $(\mathtt{P}, \mathit{same}, \mathtt{P}) \in \mathit{trust}$.

Finally, we should notice that obtaining peer consistent answers has at least the data complexity of consistent query answering, for which some results are known [7, 9, 12]. In the latter case, for common database queries and ICs, Π_2^P-completeness is easily achieved. On the other side, the problem of skeptical query evaluation from the disjunctive programs we are using for P2P systems is also Π_2^P-complete in data complexity [10]. In this sense, the logic programs are not contributing with additional complexity to our problem.

4 A LAV Approach

There are some clear connections between P2P query answering and virtual integration of data sources by means of mediator based systems [16, 21]. There are basically two approaches to the latter problem. According to *global-as-view* (GAV), each virtual table at the mediator (global) level is expressed as a view of the collection of relations in the data sources. According to *local-as-view* (LAV), relations in the (local) data sources as expressed as views of the virtual global relations. GAV is more natural and simpler for query evaluation than LAV, but LAV is simpler to deal with when sources leave and enter the integration system. GLAV is a mixture of the two approaches (see [18] for a survey).

The logic programming-based approach proposed in Section 3 can be assimilated to the GAV approach, because tables in the solutions are specified as views over peer's schemas. However, a LAV approach could also be attempted, and we also introduce virtual, global versions S_1', S_2' of S_1, S_2. The source relations R_1, R_2, S_1, S_2 are defined as views of the virtual relations in a solution, namely by $R_1(x,y) \leftarrow R_1'(x,y).$, $R_2(x,y) \leftarrow R_2'(x,y).$, $S_1(x,y) \leftarrow S_1'(x,y).$, $S_2(x,y) \leftarrow S_2'(x,y).$; and are declared closed, open, clopen and clopen, resp. [15]. These labels depend on the IC (1) and the trust relationships; actually by the fact that R_1, R_2 can change, but not S_1, S_2. More precisely, the closure of R_1 corresponds to the fact that (1) can be satisfied by deleting tuples from R_1, then the contents of the view defined in there must be contained in the original material source relation. The openness of R_2 indicates that we can insert tuples into R_2 to satisfy the constraint, and then, the extension of the solution contains the original source. Since, S_1, S_2 do not change, they are declared as both closed and open, i.e. clopen.

If a query is posed to peer P, it has to be first formulated in terms of R_1', R_2', and then it can be peer consistently answered by querying the integration system subject to the global IC: $\forall xyz \exists w (R_1'(x,y) \wedge S_1'(z,y) \rightarrow R_2'(x,w) \wedge S_2'(z,w))$. A methodology that is similar to the one applied for consistently querying virtual data integration systems under LAV can be used. In [5] methodologies for open sources are presented, and in [3] the mixed case with both open, closed and clopen sources is treated. However, there are differences with the P2P scenario; and the methodologies need to be adjusted as discussed below.

The methodology presented in [3] for CQA in virtual data integration is based on a three-layered answer set programming specification of the repairs of the system: a first layer specifies the contents of the global relations in the minimal legal instances (to this layer only open and clopen sources contribute), a second layer consisting of program denial constraints that prunes the models that violate the closure condition for the closed sources; and a third layer specifying the minimal repairs of the legal instances [5] left by the other layers wrt the global ICs. For CQA, repairs are allowed to violate the original labels.

In our P2P scenario, we want, first of all, to consider only the legal instances that satisfy the mapping in the table and that, in the case of closed sources, include the maximum amount of tuples from the sources (the virtual relations must be kept as close as possible to their original, material versions). For the kind of mappings that we have in the table, this can be achieved by using exactly the same kind of specifications presented in in [3] for the mixed case, *but* considering the closed sources as clopen. In doing so, they will contribute to the program with both rules that import their contents into the system (maximizing the set of tuples in the global relation) and denial program constraints. Now, the trust relation also makes a difference. In order for the virtual relations to satisfy the original labels, that in their turn capture the trust relationships, the rules that repair the chosen legal instances will consider only tuple deletions (insertions) for the virtual global relations corresponding to the closed (resp. open) sources. For clopen sources the rules can neither add nor delete tuples.[3] This methodology can handle universal and simple referential DECs (no cycles and single atom consequents, conditions that are imposed by the repair layer of the program), which covers a broad class of DECs. The DEC in (1) does not fall in this class, but the repair layer can be adjusted in order to generate the solutions for P.[4]

[3] This preference criterion for a subclass of the repairs is similar to the *loosely-sound semantic* for integration of open sources under GAV [17].

[4] For the specification, c.f. the appendix in http://arxiv.org/abs/cs.DB/0401015

5 The Transitive Case

It is natural to consider *transitive* DECs when a peer A, that is being queried, gets data from a peer B, who in its turn -and without A possibly knowing- gets data from a peer C to answer A's request. Most likely there is no explicit DEC from A to C; and we do not want to derive them. In order to approach peer consistent query answering in this more complex scenario, it becomes necessary to integrate the local solutions, what can be achieved by integrating the "local" specification programs. In this case, we prefer to define the global solutions directly from the the stable models of the combined program obtained from the specification of direct interactions. This is more natural and simpler than extending to the global case the definition of solution for the direct case.[5] Of course, there might be no solutions, what is reflected in the absence of stable models for the program. A problematic case appears when there are implicit cyclic dependencies [16].
Example 5. (example 4 continued) Let us consider another peer C with a relation $U(\cdot,\cdot)$. The following exchange constraint $\Sigma(\mathtt{Q},\mathtt{C}):\ \forall x \forall y(U(x,y) \rightarrow S_1(x,y))$ exists from Q to C and $(\mathtt{Q}, \mathit{less}, \mathtt{C}) \in \mathit{trust}$, meaning that Q trusts C's data more than its own. When P requests data from Q, the latter will request data from C's relation U. Now, consider the peer instances: $r_1 = \{(a,b)\}, s_1 = \{\}$, $r_2 = \{\}$, $s_2 = \{(c,e),(c,f)\}$ and $u = \{(c,b)\}$. If we analyze each peer locally, the solution for Q would contain the tuple $S_1(c,b)$ added; and P would have only one solution, corresponding to the original instances, because the DEC is satisfied without making any changes. When considering them globally, the tuple that is locally added into Q requires tuples to be added and/or deleted into/from P in order to satisfy the DEC. The combined program that specifies the global solutions consists of rules (2), (4) plus both (3) and (5), but with S_1 replaced by S_1' in the body. Finally, we add $S_1'(x,y) \leftarrow S_1(x,y),\ \mathit{not}\ \neg S_1'(x,y)$, which is a persistence rule for S_1; and $S_1'(x,y) \leftarrow U(x,y),\ \mathit{not}\ S_1(x,y)$, which enforces the satisfaction of $\Sigma(\mathtt{Q},\mathtt{C})$. The solutions obtained from the stable models are: $r' = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b), R_2'(a,f), R_1'(a,b)\}, r'' = \{S_2(c,e), S_2(c,f),$ $U(c,b), S_1'(c,b)\}$, $r''' = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b), R_2'(a,e), R_1'(a,b)\}$. $\square$

Acknowledgements. Research funded by NSERC Grant 250279-02 and a grant from the CITO/IBM-CAS Student Internship Program. L. Bertossi is Faculty Fellow of the IBM Center for Advanced Studies, Toronto Lab. We appreciate conversations with Ariel Fuxman and Pablo Barcelo.

References

1. M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM Symposium on Principles of Database Systems (PODS 99)*, ACM Press, 1999, pp. 68–79.
2. P. Barcelo, L. Bertossi, and L. Bravo. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In *Semantics in Databases*, Springer LNCS 2582, 2003, pp. 1–27.

[5] The approaches to P2P data exchange semantics in [8, 11] also appeal to this kind of 2-step process, however in a framework based on epistemic logic.

3. L. Bertossi and L. Bravo. Consistent Query Answers in Virtual Data Integration Systems. Book chapter in ‘Inconsistency Tolerance in Knowledge-bases, Databases and Software Specifications’, Springer, to appear.
4. L. Bertossi and J. Chomicki. Query Answering in Inconsistent Databases. In *Logics for Emerging Applications of Databases*, J. Chomicki, G. Saake and R. van der Meyden (eds.), Springer, 2003.
5. L. Bravo and L. Bertossi. Logic Programs for Consistently Querying Data Sources In *Proc. International Joint Conference on Artificial Intelligence (IJCAI 03)*, Morgan Kaufmann, 2003, pp. 10–15.
6. L. Bravo and L. Bertossi. Disjunctive Deductive Databases for Computing Certain and Consistent Answers to Queries from Mediated Data Integration Systems To appear in *Journal of Applied Logic*.
7. A. Cali, D. Lembo, and R. Rosati. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proc. ACM Symposium on Principles of Database Systems (PODS 03)*, ACM Press, 2003, pp. 260-271.
8. D. Calvanese, E. Damaggio, G. De Giacomo, M. Lenzerini, and R. Rosati. Semantic Data Integration in P2P Systems. In *Proc. International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 03)*, Springer LNCS 2944, 2004.
9. J. Chomicki and J. Marcinkowski. Minimal-Change Referential Integrity Maintenance Using Tuple Deletions. arXiv.org paper cs.DB/0212004. To appear in *Information and Computation.*
10. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity And Expressive Power Of Logic Programming. *ACM Computer Surveys*, 2001, 33(3), 374-425.
11. E. Franconi, G. Kuper, L. Lopatenko, L. Serafini. A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems. In *Proc. International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 03)*, Springer LNCS 2944, 2004.
12. A. Fuxman and R.J. Miller. Towards Inconsistency Management in Data Integration Systems. In *Proc. IJCAI-03 Workshop on Information Integration on the Web*.
13. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365–385.
14. F. Giannotti, D. Pedreschi, D. Sacca, C. Zaniolo. Non-Determinism in Deductive Databases. In *Proc. International Conference on Deductive and Object-Oriented Databases (DOOD 91)*, Springer LNCS 566, 1991, pp. 129–146.
15. G. Grahne and A. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. In *Proc. International Conference on Database Theory (ICDT 99)*, Springer LNCS 1540, 1999, pp. 332–347.
16. A.Y. Halevy, Z.G. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *Proceedings International Conference on Data Engineering (ICDE 03)*, 2003, pp. 505-518.
17. D. Lembo, M. Lenzerini, and R. Rosati. Source Inconsistency and Incompleteness in Data Integration. In Proc. Knowledge Representation meets Databases (KRDB 02), 2002.
18. M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. ACM Symposium on Principles of Database Systems (PODS 02)*, ACM Press, 2002, pp. 233-246.
19. N. Leone *et al.* The DLV System for Konwledge Representation and Reasoning. arXiv.org paper cs.LO/0211004. To appear in *ACM Transactions on Computational Logic*.
20. A. Levy. Logic-Based Techniques in Data Integration. In *Logic Based Artificial Intelligence*, J. Minker (ed.), Kluwer, 2000, pp. 575-595.
21. I. Tatarinov *et al.* The Piazza Peer Data Management Project. *ACM SIGMOD Record*, 2003, 32(3):47-52.