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 [1,8], 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 (a full treatment is left for an extended version of this paper). One of the approaches 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 making the DECs and the trust relationships to be satisfied.

We first recall the definition of database repair that is used to characterize the consistent answers to queries in single relational databases wrt certain integrity constraints (ICs) [1]. Given a relational database instance r with schema $\mathcal{R}$ (which includes a domain D), $\Sigma(r)$ is the set of ground atomic formulas $\{P(\bar{a}) \mid P \in \mathcal{R} \text{ and } r \models P(\bar{a})\}$.

Definition 1. [1] (a) Let r_1, r_2 be database instances over $\mathcal{R}$. The *distance*, $\Delta(r_1, r_2)$, between r_1 and r_2 is the symmetric difference $\Delta(r_1, r_2) = (\Sigma(r_1) \setminus \Sigma(r_2)) \cup (\Sigma(r_2) \setminus \Sigma(r_1))$.
(b) For database instances r, r_1, r_2, we define $r_1 \leq_r r_2$ if $\Delta(r, r_1) \subseteq \Delta(r, r_2)$.
(c) Let IC be a set of ICs on $\mathcal{R}$. A *repair* of an instance r wrt IC is a $\leq_r$-minimal instance r', such that $r' \models IC$. □
A repair of an instance r is a consistent instance that minimally differs from r.

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 2. A *P2P data exchange system* $\mathfrak{P}$ consists of:
(a) A finite set $\mathcal{P}$ of peers, denoted by $\mathtt{A}$, $\mathtt{B}$, ...
(b) For each peer $\mathtt{P}$, a database schema $\mathcal{R}(\mathtt{P})$, that includes a domain $D(\mathtt{P})$, and relations $R(\mathtt{P}), \ldots$. However, it may be natural and 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 schemata $\mathcal{R}(\mathtt{P})$ 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 *trust relation* $\mathit{trust} \subseteq \mathcal{P} \times \{\mathit{less}, \mathit{same}\} \times \mathcal{P}$, with the intended semantics that when $(\mathtt{A}, \mathit{less}, \mathtt{B}) \in \mathit{trust}$, peer $\mathtt{A}$ trusts itself less than $\mathtt{B}$; while $(\mathtt{A}, \mathit{same}, \mathtt{B}) \in \mathit{trust}$ indicates that $\mathtt{A}$ trusts itself the same as $\mathtt{B}$. In this relation, the second argument functionally depends on the other two. □
Each peer $\mathtt{P}$ is responsible for the update and maintenance of its instance wrt $IC(\mathtt{P})$, independently from other peers. In particular, we assume $r(\mathtt{P}) \models IC(\mathtt{P})$.[1]

[1] It would not be difficult to extend this scenario to one that allows local violations of ICs. Techniques as those described in [8] could be used in this direction.

Peers may submit queries to another peer in accordance with the restrictions imposed by the DECs and using the other peer's relations appearing in them.

Definition 3. (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 $\mathcal{R}(\mathtt{P})$, we denote by $\bar{r}$, the database instance on $\overline{\mathcal{R}}(\mathtt{P})$, consisting of the union of r with all the peers' instances whose schemas appear in $\overline{\mathcal{R}}(\mathtt{P})$.
(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 $\mathcal{R}(\mathtt{P}) \subseteq \mathcal{S}$, then $r|\mathtt{P}$ denotes the restriction of r to $\mathcal{R}(\mathtt{P})$.
(d) We denote by $\mathcal{R}(\mathtt{P})^{less}$ the union of all schemata $\mathcal{R}(\mathtt{Q})$, with $(\mathtt{P}, less, \mathtt{Q}) \in$ *trust*. Analogously is $\mathcal{R}(\mathtt{P})^{same}$ defined. □

From the perspective of a peer $\mathtt{P}$, its own database may be inconsistent wrt the data owned by another peer $\mathtt{Q}$ and the DECs in $\Sigma(\mathtt{P},\mathtt{Q})$. Only when $\mathtt{P}$ trust $\mathtt{Q}$ the same as or more than itself, it has to consider $\mathtt{Q}$'s data. When $\mathtt{P}$ queries its database, these inconsistencies may have to be taken into account. Ideally, the answers to the query obtained from $\mathtt{P}$ should be consistent with $\Sigma(\mathtt{P},\mathtt{Q})$ (and its own ICs $\Sigma(\mathtt{P})$). In principle, $\mathtt{P}$, who is not allowed to change other peers' data, could try to repair its database in order to satisfy $\Sigma(\mathtt{P}) \cup IC(\mathtt{P})$. This is not a realistic approach. Rather $\mathtt{P}$ should solve its conflicts 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 $\mathtt{P}$ 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 $\mathtt{P}$'s trust relationships to other peers. This repair process may lead to alternative global databases called the *solutions* for $\mathtt{P}$. Next, the peer consistent answers from $\mathtt{P}$ 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 $\mathtt{P}$, those whose relations appear in its trusted exchange constraints, and are trusted by $\mathtt{P}$ 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 4.3).

Definition 4. (direct case) Given a peer $\mathtt{P}$ in a P2P data exchange system $\mathfrak{P}$ and an instance r on $\mathcal{R}$, we say that an instance r' on $\mathcal{R}$ is a *solution for* $\mathtt{P}$ if, simultaneously: (a) $r' \models \Sigma(\mathtt{P}) \cup IC(\mathtt{P})$. (b) $r'|P = r|P$ for every predicate $P \notin \overline{\mathcal{R}}(\mathtt{P})$. (c) There are instances r_1, r_2 over $\mathcal{R}$ satisfying: (c1) $r_2 = r'$. (c2) r_1 is a repair of r wrt $\bigcup\{\Sigma(\mathtt{P},\mathtt{Q}) \mid (\mathtt{P}, less, \mathtt{Q}) \in \textit{trust}\}$, with $r_1|\mathtt{Q} = r|\mathtt{Q}$ whenever $(\mathtt{P}, less, \mathtt{Q}) \in \textit{trust}$ or $(\mathtt{P}, same, \mathtt{Q}) \in \textit{trust}$. (c3) r_2 is a repair of r_1 wrt $\bigcup\{\Sigma(\mathtt{P},\mathtt{Q}) \mid (\mathtt{P}, same, \mathtt{Q}) \in \textit{trust}\}$, such that $r_2 \models \Sigma(\mathtt{P},\mathtt{Q})$ and $r_2|\mathtt{Q} = r_1|\mathtt{Q}$ for those peers $\mathtt{Q}$ with $(\mathtt{P}, less, \mathtt{Q}) \in \textit{trust}$. □

The *solutions* for a peer are used as a conceptual, auxiliary tool to characterize the semantically correct answers to a peer's queries. We are not interested in computing a peer's solutions *per se*. Solutions (and repairs) 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 intuitive terms, a solution for `P` repairs the global instance, but leaves unchanged the tables that do not appear in its trusted ICs and those tables that belong to peers that are more trusted by him than himself. With this condition, `P` first tries to change its own tables according to what the dependencies to more trusted peers of peers prescribe. Next, keeping those more trusted dependencies satisfied, it tries to repair its or other peers' data, but only considering those peers who are equality trusted as itself.

In these definitions we find clear similarities with the characterization of consistent query answers in single relational databases [8]. 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 (other important differences are discussed below).

Example 1. Consider a P2P data system with peers `P1`, `P2`, `P3`, and schemas $\mathcal{R}_i = \{R^i, \ldots\}$, and instances r^i, $i = 1, 2, 3$, resp.; and: (a) $r^1 = \{R^1(a, b), R^1(s, t)\}$, $r^2 = \{R^2(c, d), R^2(a, e)\}$, $r^3 = \{R^3(a, f), R^3(s, u)\}$. (b) $\mathit{trust} = \{$ $(\mathtt{P1}, \mathit{less}, \mathtt{P2})$, $(\mathtt{P1}, \mathit{same}, \mathtt{P3})$ $\}$. (c) $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\ \forall xy(R^2(x, y) \rightarrow R^1(x, y))\ \}$; $\Sigma(\mathtt{P1}, \mathtt{P3}) = \{\ \forall xyz(R^1(x, y) \wedge R^3(x, z) \quad \rightarrow \quad y = z)\ \}$.

Here, the global instance is $r = \{R^1(a, b), R^1(s, t), R^2(c, d), R^2(a, e), R^3(a, f), R^3(s, u)\}$. The solutions for `P1` are obtained by first repairing r wrt the relationship between `P1` and `P2`. Then r_1 in condition (c2) in Definition 4 is $r_1 = \{R^1(a, b), R^1(s, t),\ R^1(c, d), R^1(a, e),\ R^2(c, d), R^2(a, e),\ R^3(a, f), R^3(s, u)\}$. In this example there is only one repair at this stage, but in other situations there might be several. Now, this repair has to be repaired in its turn wrt the data dependency between `P1` and `P3` (but keeping the relationship between `P1` and `P2` satisfied). In this case, we obtain only two repairs, $r' = \{R^1(a, b), R^1(s, t), R^1(c, d), R^1(a, e), R^2(c, d), R^2(a, e)\}$; and $r'' = \{\ R^1(a, b), R^1(c, d), R^1(a, e), R^2(c, d), R^2(a, e), R^3(s, u)\}$. These are the only solutions for peer `P1`. $\square$

The minimization involved in a solution is similar to a prioritized minimization (with some predicates that are kept fixed) found in non-monotonic reasoning [25]. Actually, the notion of consistent query answer -even the one based on the non prioritized version of repair (c.f. Definition 1)- is a non-monotonic notion [8].[2]

Notice that the notion of a solution for a peer `P` is a "local notion" in the sense that it considers the "direct neighbors" of `P` only. One reason for considering this case is that `P` does not see beyond its neighbors; and when `P` requests data to a neighbor, say `Q`, the latter may decide -or even `P` may decide a priori and in a uniform way- that for `P` it is good enough to accommodate its data to its neighbors alone, without considering any transitive dependencies. In section 4.3 we will explore the case of interrelated dependencies.

Now we can define which are the intended answers to a query posed to a peer, from the perspective of that peer.

[2] A circumscriptive approach to database repairs was given in [9]. It should not be difficult to extend that characterization to capture the peer solutions.

Definition 5. Given a FO query $Q(\bar{x}) \in \mathcal{L}(\mathtt{P})$, posed to peer $\mathtt{P}$, a ground tuple $\bar{t}$ is *peer consistent* for $\mathtt{P}$ iff $r'|\mathtt{P} \models Q(\bar{t})$ for every solution r' for $\mathtt{P}$. $\Box$

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 notion of solution for a single peer.

Peer consistent answers to queries can be obtained by using techniques similar to those developed for CQA, for example, query rewriting based techniques [1,8]. However, there are important differences, because now we have some fixed predicates in the repair process.

Example 2. (example 1 continued) If $\mathtt{P1}$ is posed the query $Q\colon R^1(x,y)$ asking for the tuples in relation R^1, we first rewrite the query by considering the exchange dependencies in $\Sigma(\mathtt{P1},\mathtt{P2})$, obtaining $Q'\colon R^1(x,y) \vee R^2(x,y)$, which basically has the effect of bringing $\mathtt{P2}$'s data into $\mathtt{P1}$. Next, the exchange dependency $\Sigma(\mathtt{P1},\mathtt{P3})$ is considered, and now the query is rewritten into

$$Q''\colon [R^1(x,y) \wedge \forall z_1(R^3(x,z_1) \wedge \neg\exists z_2 R^2(x,z_2) \ \rightarrow \ z_1 = y)] \ \ \vee \ \ R^2(x,y). \quad (1)$$

In order to answer this query, $\mathtt{P1}$ will first issue a query to $\mathtt{P2}$ to retrieve the tuples in R^2; next, a query is issued to $\mathtt{P3}$ to leave outside R^1 those tuples that appear with the same first but not the same second argument in R^3, as long as the conflicting tuple in R^2 is "protected" by a tuple in R^3 which has the same key as a the two conflicting tuples in R^1 and R^3 ($R^1(a,b)$ above). The answers to query (1) are $(a,b), (c,d), (a,e)$, precisely the peer consistent answers to query Q for peer $\mathtt{P1}$ according to their semantic definition. $\Box$

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. This makes sense, because the peer may import data from other peers. This is another difference with CQA, where all consistent answers are answers to the original query[3].

The query rewriting approach suggested in Example 2 differs from the one used for CQA. In the latter case, literals in the query are resolved (using *resolution*) against the ICs in order to generate residues that are appended as extra conditions to the query, in an iterative process. 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 -as in the query rewriting based consistent query answering- but instead, by relaxing the query in some sense.

Instead of pursuing and fully developing a FO query transformation approach to query answering in P2P systems, we will propose (see Section 3) an alternative methodology based on answer set programming, which is more general. Furthermore, since query answering in P2P systems already includes some sufficiently complex cases of CQA, a FO query rewriting approach to P2P query answering is bound to have important limitations in terms of completeness, as in CQA [8]; for example in the case of existential queries and/or existential DECs.

[3] At least if the ICs are *generic* [8], i.e. they do not imply by themselves the presence/absence of any particular ground tuple in/from the database.

3 Referential Exchange Constraints

In most applications we may expect the exchange constraints to be inclusion dependencies or referential constraints, i.e. formulas of the form

$$\forall \bar{x} \exists \bar{y} (R^{\mathtt{Q}}(\bar{x}) \wedge \cdots \quad \rightarrow R^{\mathtt{P}}(\bar{z}, \bar{y}) \wedge \cdots), \qquad (2)$$

where $R^{\mathtt{Q}}, R^{\mathtt{P}}$ are relations for peers Q and P, resp., the dots indicate some possible additional conditions, most likely expressed in terms of built-ins, $\bar{z} \subseteq \bar{x}$ (if $\bar{y} = \emptyset$ and $\bar{z} = \bar{x}$, and no additional conditions are given, we have a full inclusion dependency, like $\Sigma(\mathtt{P1}, \mathtt{P2})$ in Example 1).

An exchange constraint of the form (2) will most likely belong to $\Sigma(\mathtt{P}, \mathtt{Q})$, i.e. to peer P, who wants to import data from the more trustable peer Q. It could also belong to Q, if this peer wants to validate its own data against the data at P's site. Section 3.1 shows an example of a more involved referential constraint.

An answer set programming approach to the specification of solutions for a peer can be developed. In spirit, those specifications would be similar to those of repairs of single relational databases under referential integrity constraints [3]. However -as already seen in Examples 1 and 2- there are important differences with CQA. In Section 3.1 we give an example that shows the main issues around this kind specification.[4]

3.1 An extended example

Consider a P2P data exchange system with peers P and Q, with schemas $\{R_1(\cdot, \cdot), R_2(\cdot, \cdot)\}$, $\{S_1(\cdot, \cdot), S_2(\cdot, \cdot)\}$, resp. Peer P also has the exchange constraint

$$\forall x \forall y \forall z \exists w (R_1(x, y) \wedge S_1(z, y) \ \rightarrow\ R_2(x, w) \wedge S_2(z, w)), \qquad (3)$$

which mixes tables of the two peers on each side of the implication.

Let us assume that peer P is querying his database, but subject to its DEC (3). We will consider the case where $(\mathtt{P}, \mathit{less}, \mathtt{Q}) \in \mathit{trust}$, i.e. P considers Q's data more reliable than his own. If (3) is satisfied by the combination of the data in P and Q, then the current global instance constitutes P's solution. Otherwise, alternative solutions for P have to be found, keeping Q's data fixed in the process. This is the case, where there are ground tuples $R_1(d, m) \in r(\mathtt{P}), S_2(a, m) \in r(\mathtt{Q})$, 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 to queries for peer P amounts to virtually restoring the satisfaction of (3), actually by virtually modifying P's data. In order to specify P's modified relations, we introduce virtual versions R_1', R_2' of R_1, R_2, which will contain the data in peer P's solutions. In consequence, at the solution level, we have the relations R_1', R_2', S_1, S_2. Since P is querying its database, its original queries will be expressed in terms of relations R_1', R_2' only (plus possible built-ins).

The contents of the virtual relations R_1', R_2' will be obtained from the contents of the material sources R_1, R_2, S_1, S_2.[5] Since S_1, S_2 are fixed, the satisfaction of

[4] A detailed and complete approach will be found in an extended version of this paper.

[5] We can observe that the virtual relations can be seen as virtual global relations in a virtual data integration system [24,21]. For a more detailed comparison between data integration and peer data management systems see [19,26].

(3) requires R_1' to be a subset of R_1, and R_2', a superset of R_2. The specification of these relations can be done in disjunctive extended logic programs with answer set (stable model) semantics [16]. The first rules for the specification program Π are:

$$R_1'(x,y) \leftarrow R_1(x,y),\ \mathit{not}\ \neg R_1'(x,y) \tag{4}$$
$$R_2'(x,y) \leftarrow R_2(x,y),\ \mathit{not}\ \neg R_2'(x,y), \tag{5}$$

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 (3) (with R_1, R_2 replaced by R_1', R_2'). Some of the exceptions for R_1' are specified by

$$\neg R_1'(x,y) \leftarrow R_1(x,y), S_1(z,y),\ \mathit{not}\ \mathit{aux}_1(x,z),\ \mathit{not}\ \mathit{aux}_2(z) \tag{6}$$
$$\mathit{aux}_1(x,z) \leftarrow R_2(x,w), S_2(z,w) \tag{7}$$
$$\mathit{aux}_2(z) \leftarrow S_2(z,w). \tag{8}$$

That is, $R_1(x,y)$ is deleted if it participates in a violation of (3) (what is captured by the first three literals in the body of (6) plus rule (7)), 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 (6) plus rule (8)). In case there is such a tuple in S_2, then we have the alternative of either deleting a tuple from R_1 or inserting a tuple into R_2:

$$\neg R_1'(x,y) \vee R_2'(x,w) \leftarrow R_1(x,y), S_1(z,y),\ \mathit{not}\ \mathit{aux}_1(x,z), S_2(z,w), \mathit{choice}((x,z),w). \tag{9}$$

That is, in case of a violation of (3), when there is tuple of the form (a,t) in S_2 for the combination of values (d,a), then the *choice operator* [17] 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. Notice that no exceptions are specified for R_2', what makes sense since R_2' is a superset of R_2. In consequence, the negative literal in the body of (5) can be eliminated. However, new tuples can be inserted into R_2', what is captured by rule (9). Finally, the program must contain as facts the tuples in the original relations R_1, R_2, S_1, S_2.

In the case where P equally trusts himself and Q, both P and Qs' relations become flexible when searching for a solution for P. The program becomes more involved, because now S_1, S_2 may also change. In consequence, virtual versions for them should be introduced and specified.

3.2 Considerations on specifications of peers' solutions

The example we presented in Section 3.1 shows the main issues in the specification of a peer's solutions under referential exchange constraints. If desired, the choice operator can be replaced by a predicate that can be defined by means of extra rules, producing the so-called stable version of the choice program [17]. This stable version has a completely standard answer set semantics.

The peer's solutions are in one to one correspondence with the answer sets of the program. In the previous example, each solution r^S for peer P coincides with the original, material, global instance for the tables other than R_1, R_2, whereas the contents r_1^S, r_2^S for these two are of the form $r_i^S = \{\bar{t} \mid R_i'(\bar{t}) \in S\}$, where S is an answer set of program Π. The absence of solutions for a peer will thus be captured by the non existence of answer sets for program Π.

Program Π represents in a compact form all the solutions for a peer; in consequence, the peer consistent answers to a query posed to the peer can be obtained by running the query, expressed as a query program in terms of the virtually repaired tables, in combination with the specification program Π. The answers so obtained will be those that hold for all the possible solutions if the program is run under the skeptical answer set semantics. As for consistent query answering, a system like DLV [14] can be used for this purpose.

For example, the query $Q(x, z): \ \exists y(R_1(x, y) \wedge R_2(z, y))$ issued to peer P, would be peer consistently answered by running the query program $Ans_Q(x, z) \leftarrow R_1'(x, y), R_2'(x, y)$ together with program Π. Although only (the new versions of) P's relations appear in the query, the program may make P import data from Q.

If a peer has local ICs that have to be satisfied and a program has been used to specify its solutions, then the program should take care of those constraints. One simple way of doing this consists in using program denial constraints. If in Section 3.1 we had for peer 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 $\leftarrow R_1(x, y), R(x, z), y \neq z$, which would have the effect of pruning those solutions (or models of the program) that do not satisfy the FD. DLV, for example, can handle program denial constraints [23].

A more flexible alternative to 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 [3].

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 [12,15,11]. 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 [13]. In this sense, the logic programs are not contributing with additional complexity to our problem.

4 Discussion and Extensions

4.1 Optimizations

It is possible to perform some optimizations on the program, to make its evaluation simpler. Disjunctive program under the stable model semantics are more complex than non disjunctive programs [13]. However, it is known that a disjunctive program can be transformed into a non disjunctive program if the program is head-cycle free (HCF) [4,22]. Intuitively speaking, a disjunctive program is HCF if there are no cycles involving two literals in the head of a same rule, where a link is established from a literal to another if the former appears positive in the body of a rule, and the latter appears in the head of the same rule. These considerations about HCF programs hold for programs that do not contain the choice operator, i.e. they might not automatically apply to our programs that specifies the solutions for a peer under referential constraints. However, it is possible to prove that a disjunctive choice program Π is HCF when the program obtained from Π by removing its choice goals is HCF. [6].

Example 3. Consider the choice program Π presented in Section 3.1. If the choice operator is eliminated from rule (9), we are left with the rule

$$\neg R_1'(x,y) \vee R_2'(x,w) \leftarrow R_1(x,y), S_1(z,y),\ \mathit{not}\ \mathit{aux}_1(x,z), S_2(z,w).$$

The resulting program is HCF and then rule (9) can be replaced by two rules:

$$\begin{aligned}\neg R_1'(x,y) \leftarrow{} & R_1(x,y), S_1(z,y),\ \mathit{not}\ \mathit{aux}_1(x,z), S_2(z,w),\ \mathit{not}\ R_2'(x,w),\\ & \mathit{choice}((x,z),w).\\ R_2'(x,w) \leftarrow{} & R_1(x,y), S_1(z,y),\ \mathit{not}\ \mathit{aux}_1(x,z), S_2(z,w),\ \mathit{not}\ \neg R_1'(x,y),\\ & \mathit{choice}((x,z),w).\end{aligned}$$

□

4.2 A LAV approach

The logic programming-based approach proposed in Section 3.1 can be seen assimilated to the global-as-view (GAV) approach to virtual data integration [21], in the sense that the tables in the solutions are specified as views over the peer's schemas. However, a local-as-view (LAV) approach could also be attempted. In this case, we also introduce virtual, global versions of S_1, S_2. The relations in the sources have to be defined as views of the virtual relations in a solution, actually, through the following specification of a virtual integration system [18]

View definitions	label	source
$R_1(x,y) \leftarrow R_1'(x,y)$	*closed*	r_1
$R_2(x,y) \leftarrow R_2'(x,y)$	*open*	r_2
$S_1(x,y) \leftarrow S_1'(x,y)$	*clopen*	s_1
$S_2(x,y) \leftarrow S_2'(x,y)$	*clopen*	s_2

Here the r_i, s_j are the original material extensions of relations R_i, S_j. The labels for the sources are assigned on the basis of the view definitions in the first column, the IC (3) and the trust relationships; in the latter case, by the fact that R_1, R_2 can change, but not S_1, S_2. More precisely, the label in the first row corresponds to the fact that (3) can be satisfied by deleting tuples from R_1, then the contents of the view defined in there must be contained in the original relation r_1 (the material source). The label in the second row indicates that we can insert tuples into R_2 to satisfy the constraint, and then, the extension of the solution contains the original source r_2. Since, S_1, S_2 do not change, we declare them as both closed and open, i.e. clopen.

If a query is posed to, say 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: $\forall x \forall y \forall z \exists w (R_1'(x,y) \wedge S_1'(z,y) \rightarrow R_2'(x,w) \wedge S_2'(z,w))$. A methodology that is similar to the one applied for consistently querying virtual data integration systems under LAV can be used. In [7,10] methodologies for open sources are presented, and in [5] the mixed case with both open, closed and clopen sources is treated. However, there are differences in our P2P scenario; and those methodologies need to be adjusted.

The methodology presented in [5] 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 [7] 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 [5] 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.[6] 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 (3) does not fall in this class, but the repair layer can be easily adjusted in order to generate the solutions for peer P. Due to space limitations, the program is given in the appendix.

4.3 Beyond direct solutions

It is natural to considers *transitive data exchange dependencies*. This is a situation that arises when, e.g. a peer A, that is being queried, gets data from another peer B, who in its turn -and without A possibly knowing- gets data from a third peer C to answer A's request. Most likely there won't be any explicit DEC from A to C capturing this transitive exchange; and we do not want to derive any.

In order to attack 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, it is much more natural and simpler than extending the definition of solutions for the direct case, to define the semantics of a peer's (global) solutions directly as the answer

[6] This preference criterion for a subclass of the repairs is similar to the *loosely-sound semantic* for integration of open sources under GAV [20].

sets of the combined programs. 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 [19].

Example 4. (example in Section 3.1 continued) Let us consider another peer C. 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 (4), (5),(7), (8) plus ((10), (11) replace (6), (9), resp.)

$$\neg R_1'(x,y) \leftarrow R_1(x,y), S_1'(z,y), \ \mathit{not} \ \mathit{aux}_1(x,z), \ \mathit{not} \ \mathit{aux}_2(z) \quad (10)$$

$$\neg R_1'(x,y) \vee R_2'(x,w) \leftarrow R_1(x,y), S_1'(z,y), \ \mathit{not} \ \mathit{aux}_1(x,z), S_2(z,w), \mathit{choice}((x,z),w) \quad (11)$$

$$S_1'(x,y) \leftarrow S_1(x,y), \ \mathit{not} \ \neg S_1'(x,y) \quad (12)$$

$$S_1'(x,y) \leftarrow U(x,y), \ \mathit{not} \ S_1(x,y). \quad (13)$$

The solutions obtained from the stable models of the program are precisely the expected ones: $r_1 = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b), R_2'(a,f), R_1'(a,b)\}$, $r_2 = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b)\}$, $r_3 = \{S_2(c,e), S_2(c,f), U(c,b), S_1'(c,b), R_2'(a,e), R_1'(a,b)\}$. □

Acknowledgements: Research funded by NSERC Grant 250279-02. L. Bertossi is Faculty Fellow of the IBM Center for Advanced Studies, Toronto Lab. We are grateful for stimulating conversations with Ariel Fuxman, who, in particular, proposed to consider the example in Section 3.1.

References

1. Arenas, M., Bertossi, L., Chomicki, J. Consistent Query Answers in Inconsistent Databases. Proc. ACM Symposium on Principles of Database Systems (PODS 99), 1999, pp. 68–79.
2. Arenas, M., Bertossi, L., Chomicki, J. Answer Sets for Consistent Query Answers. *Theory and Practice of Logic Programming*, 3, 4&5, 2003, pp. 393-424.
3. Barcelo, P., Bertossi, Bravo, L. 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.
4. Ben-Eliyahu, R. and Dechter, R. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics in Artificial Intelligence*, 1994, 12:53-87.
5. Bertossi, L. and Bravo, L. Consistent Query Answers in Virtual Data Integration Sistems. To appear as a book chapter in 'Inconsistency Tolerance in Knowledge-bases, Databases and Software Specifications'. Springer.

6. Bertossi, L. and Bravo, L. On Theoretical Aspects of Consistent Query Answering. In preparation, 2004.
7. Bertossi, L., Chomicki, J., Cortes, A. and Gutierrez, C. Consistent Answers from Integrated Data Sources. In Proc. Flexible Query Answering Systems (FQAS 02), Springer LNAI 2522, 2002, pp. 71–85.
8. Bertossi, L.; and Chomicki, J. Query Answering in Inconsistent Databases. In *Logics for Emerging Applications of Databases*, J. Chomicki, G. Saake and R. van der Meyden (eds.), Springer, 2003.
9. Bertossi, L., Schwind, C. Database Repairs and Analytic Tableaux. *Annals of Mathematics and Artificial Intelligence*, 2004, 40(1-2): 5-35.
10. Bravo, L. and Bertossi, L. Logic Programs for Consistently Querying Data Sources In Proc. of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 03), Morgan Kaufmann, 2003, pp. 10–15.
11. Cali, A., Lembo, D. and Rosati, R. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proc. ACM PODS 03*, 2003, pp. 260-271.
12. Chomicki, J. and Marcinkowski, J. Minimal-Change Referential Integrity Maintenance Using Tuple Deletions. arXiv:cs.DB/0212004.
13. Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity And Expressive Power Of Logic Programming. *ACM Computer Surveys*. 2001, 33(3), 374-425.
14. Eiter, T., Faber, W.; Leone, N., Pfeifer, G. Declarative Problem-Solving in DLV. In *Logic-Based Artificial Intelligence*. J. Minker (ed.), Kluwer, 2000, pp. 79-103.
15. Fuxman, A. and Miller, R.J. Towards Inconsistency Management in Data Integration Systems. In on-line *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web*.
16. Gelfond, M. and Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365–385.
17. Giannotti, F.; Pedreschi, D.; Sacca, D., Zaniolo, C. Non-Determinism in Deductive Databases. In Proc. DOOD, Springer LNCS 566, 1991, pp. 129–146.
18. Grahne, G. and Mendelzon, A. 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.
19. Halevy, A.Y., Ives, Z.G., Suciu, D. and Tatarinov, I. Schema Mediation in Peer Data Management Systems. Proceedings International Conference on Data Engineering (ICDE 03), 2003, pp. 505-518.
20. Lembo, D., Lenzerini, M. and Rosati, R. Source Inconsistency and Incompleteness in Data Integration. In *Proc. International Workshop Knowledge Representation meets Databases (KRDB 02)*, 2002.
21. Lenzerini, M. Data Integration: A Theoretical Perspective. In Proc. ACM Symposium on Principles of Database Systems (PODS 02), 2002, pp. 233-246.
22. Leone, N., Rullo, P. and Scarcello, F. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation*, 1997, 135(2):69-112.
23. Leone, N. et al. The DLV System for Konwledge Representation and Reasoning. arXiv:cs.AI/0211004. To appear in *ACM Transactions on Computational Logic.*
24. Levy, A. Logic-Based Techniques in Data Integration. In *Logic Based Artificial Intelligence*, J. Minker (ed.), Kluwer, 2000, pp. 575-595.
25. Lifschitz, V. Computing Circumscription. Proc. AAAI 1987.
26. Tatarinov, I. *et al.* The Piazza Peer Data Management Project. *SIGMOD Record*, 2003, 32(3):47-52.

5 Appendix:

The following answer set program specifies the solutions for the example in Section 3.1 following a LAV approach to P2P data exchange (see Section 4.2). Assume that the peers have the following instances: $r_1 = \{(a,b)\}, s_1 = \{(c,b)\}$, $r_2 = \{\}$ and $s_2 = \{(c,e),(c,f)\}$. Then, the facts of the program are: $R_1(a,b)$, $S_1(c,b), S_2(c,e), S_2(c,f)$. The layer that specifies the preferred legal instances contains the following rules:

$$\begin{aligned}
R_1'(X,Y,t_d) &\leftarrow R_1(X,Y).\\
S_1'(X,Y,t_d) &\leftarrow S_1(X,Y).\\
R_2'(X,Y,t_d) &\leftarrow R_2(X,Y).\\
S_2'(X,Y,t_d) &\leftarrow S_2(X,Y).\\
&\leftarrow R_1'(X,Y,t_d), R_1(X,Y).\\
&\leftarrow S_1'(X,Y,t_d), S_1(X,Y).\\
&\leftarrow S_2'(X,Y,t_d), S_2(X,Y).
\end{aligned}$$

The layer that specifies the repairs of the legal instances contains the following rules. The annotation constants in the third arguments in the relations are used as auxiliary elements in the repairs process [3]. The choice operator has been unfolded, producing the stable version of the choice program.

$$\begin{aligned}
R_1'(X,Y,t_{ss}) &\leftarrow R_1'(X,Y,t_d),\ \mathit{not}\ R_1'(X,Y,f_a).\\
R_1'(X,Y,t_{ss}) &\leftarrow R_1'(X,Y,t_a).\\
&\leftarrow R_1'(X,Y,t_a), R_1'(X,Y,f_a).\\
S_1'(X,Y,t_{ss}) &\leftarrow S_1'(X,Y,t_d),\ \mathit{not}\ S_1'(X,Y,f_a).\\
S_1'(X,Y,t_{ss}) &\leftarrow S_1'(X,Y,t_a).\\
&\leftarrow S_1'(X,Y,t_a), S_1'(X,Y,f_a).\\
R_2'(X,Y,t_{ss}) &\leftarrow R_2'(X,Y,t_d),\ \mathit{not}\ R_2'(X,Y,f_a).\\
R_2'(X,Y,t_{ss}) &\leftarrow R_2'(X,Y,t_a).\\
&\leftarrow R_2'(X,Y,t_a), R_2'(X,Y,f_a).\\
S_2'(X,Y,t_{ss}) &\leftarrow S_2'(X,Y,t_d),\ \mathit{not}\ S_2'(X,Y,f_a).\\
S_2'(X,Y,t_{ss}) &\leftarrow S_2'(X,Y,t_a).\\
&\leftarrow S_2'(X,Y,t_a), S_2'(X,Y,f_a).\\
R_1'(X,X,f_a) &\leftarrow R_1'(X,Y,t_d), S_1'(Z,Y,t_d),\ \mathit{not}\ \mathit{aux}_1(X,Z),\\
&\qquad \mathit{not}\ \mathit{aux}_2(Z).\\
\mathit{aux}_1(X,Z) &\leftarrow R_2'(X,U,t_d), S_2'(Z,U,t_d).\\
\mathit{aux}_2(Z) &\leftarrow S_2'(Z,W,t_d).\\
R_1'(X,Y,f_a) \vee R_2'(X,W,t_a) &\leftarrow R_1'(X,Y,t_d), S_1'(Z,Y,t_d),\ \mathit{not}\ \mathit{aux}_1(X,Z),\\
&\quad S_2'(Z,W,t_d), \mathit{chosen}(X,Z,W).\\
\mathit{chosen}(X,Z,W) &\leftarrow R_1'(X,Y,t_d), S_1'(Z,Y,t_d),\ \mathit{not}\ \mathit{aux}_1(X,Z),\\
&\quad S_2'(Z,W,t_d),\ \mathit{not}\ \mathit{diffchoice}(X,Z,W).
\end{aligned}$$

$$\mathit{diffchoice}(X, Z, W) \leftarrow \mathit{chosen}(X, Z, U), S_2'(Z, W, t_d), U \neq W.$$

The following are the stable models of the program:

$M_1 = \{R_1(a,b),\ S_1(c,b),\ S_2(c,e),\ S_2(c,f),\ R_1'(a,b,td),\ S_1'(c,b,td),\ S_2'(c,e,td),$
$S_2'(c,f,td),\ aux_2(c),\ S_1'(c,b,tss),\ S_2'(c,e,tss),\ S_2'(c,f,tss),\ R_1'(a,b,tss),$
$\mathit{diffchoice}(a,c,e),\ \mathit{chosen}(a,c,f),\ R_2'(a,f,ta),\ R_2'(a,f,tss)\}$
$M_2 = \{R_1(a,b),\ S_1(c,b),\ S_2(c,e),\ S_2(c,f),\ R_1'(a,b,td),\ S_1'(c,b,td),\ S_2'(c,e,td),$
$S_2'(c,f,td),\ aux_2(c),\ S_1'(c,b,tss),\ S_2'(c,e,tss),\ S_2'(c,f,tss),\ R_1'(a,b,fa),$
$\mathit{diffchoice}(a,c,e),\ \mathit{chosen}(a,c,f)\}$
$M_3 = \{R_1(a,b),\ S_1(c,b),\ S_2(c,e),\ S_2(c,f),\ R_1'(a,b,td),\ S_1'(c,b,td),\ S_2'(c,e,td),$
$S_2'(c,f,td),\ aux_2(c),\ S_1'(c,b,tss),\ S_2'(c,e,tss),\ S_2'(c,f,tss),\ R_1'(a,b,tss),$
$\mathit{chosen}(a,c,e),\ \mathit{diffchoice}(a,c,f),\ R_2'(a,e,ta),\ R_2'(a,e,tss)\}$
$M_4 = \{R_1(a,b),\ S_1(c,b),\ S_2(c,e),\ S_2(c,f),\ R_1'(a,b,td),\ S_1'(c,b,td),\ S_2'(c,e,td),$
$S_2'(c,f,td),\ aux_2(c),\ S_1'(c,b,tss),\ S_2'(c,e,tss),\ S_2'(c,f,tss),\ R_1'(a,b,fa),$
$\mathit{chosen}(a,c,e),\ \mathit{diffchoice}(a,c,f)\}$,

which correspond to the following solutions (they can be obtained by selecting only the tuples with annotation t_{ss}): $r^{M_1} = \{S_1'(c,b),\ S_2'(c,e),\ S_2'(c,f),\ R_1'(a,b),\ R_2'(a,f)\}$, $r^{M_2} = \{S_1'(c,b),\ S_2'(c,e),\ S_2'(c,f)\}$, $r^{M_3} = \{S_1'(c,b),\ S_2'(c,e),\ S_2'(c,f),\ R_1'(a,b),\ R_2'(a,e)\}$, $r^{M_4} = \{S_1'(c,b),\ S_2'(c,e),\ S_2'(c,f)\}$.