Information Sharing Agents in a Peer Data Exchange System

Leopoldo Bertossi[1] and Loreto Bravo[2]

[1] Carleton University, School of Computer Science, Ottawa, Canada
bertossi@scs.carleton.ca
[2] University of Edinburgh, School of Informatics, Edinburgh, UK
lbravo@inf.ed.ac.uk

Abstract. We present a semantics and answer set programs for relational *peer data exchange systems*. When a peer answers a query, it exchanges data with other peers in order to supplement or modify its own data source. The data exchange relationships between peers are specified by logical sentences called *data exchange constraints* and *trust relationships*, which together determine how data is moved around (in order to keep them satisfied). This process determines virtual, alternative instances for a peer that can be specified as the models of an answer set program. The *peer consistent answers* to a query that are returned by a peer are those that are invariant under all these instances. The logic program can be used to compute peer consistent answers.

1 Introduction

A peer data exchange system (PDES) can be seen as a set of information agents, each of them being the owner of a data source. When one of them receives a query, in order to answer it, its data is completed or modified according to relevant data that the other agents may have. More precisely, a peer data exchange system (PDES) is a finite set $\mathcal{P} = \{\mathtt{P}_1, \ldots \mathtt{P}_n\}$ of peers, each of them with a local relational database instance. A peer P may be directly related to another peer P' by means of a set $\Sigma(\mathtt{P}, \mathtt{P'})$ of *data exchange constraints* (DECs), which are first-order sentences expressed in terms of the two participating database schemas. DECs between two peers are expected to be satisfied by the combination of the two local instances.[1] However, this condition is taken into account only when local queries are answered. That is, each peer will not update its physical instance according to its DECs and other peers' instances. Instead, if a peer P is answering a query, it may, at query time import data from other peers to complement its data and/or ignore part of its own data. In which way a peer uses the data from other peers depends on its DECs, the peers' instances, and its *trust relationships* to other peers: a peer P may trust its data the same as or less than other peers' data.

In this paper we present in simple terms and by means of examples a formal semantics for such a system of peers who exchange data for query answering. The

[1] For simplicity, but without loss of generality, local schemas are pairwise disjoint.

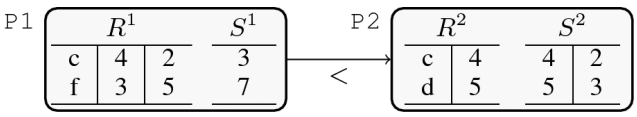
A. Hameurlain (Ed.): Globe 2008, LNCS 5187, pp. 70–81, 2008.

most important role of a semantics in this case is to characterize in precise terms what are the intended and correct answers to a query posed to and answered by a peer in the system. We propose a *model-theoretic semantics*, that is a collection of possible and admissible models over which the system is interpreted. The expected answers from a peer to a query are those that are *certain* wrt a set of database instances associated to that peer. Furthermore, our declarative semantics can be made executable, by using logic programs with stable model semantics [12] to specify the intended models. The precise formal semantics was presented in detailed technical terms in [3].

If for peer P it holds $\Sigma(\mathtt{P}, \mathtt{P'}) \neq \emptyset$, i.e. there are DECs from P to P', we say that P' is a *neighbor* of P. Clearly, DECs for a peer P can be *inconsistent* wrt (not satisfied by) the combination of its instance and those of its neighbors. A *virtual* combined instance for P that solves these inconsistencies by performing a minimal set of changes on the database relations is called a *neighborhood solution instance* for P. By restricting it to the schema of P, we get a *solution instance* for P. There might be more than one solution instance for a peer, and all of them are taken into consideration when answering queries posed to P: The *peer consistent answers* from P are those that are shared (or returned) by all the different solution instances. That is, a cautions (a.k.a. skeptical or certain) semantics is applied to query answering.

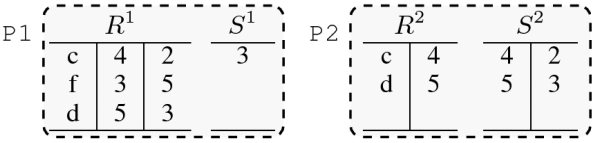
Each peer P can be seen as an ontology consisting of the database instance plus metadata that describes the database schema, local integrity constraints (ICs), its set $\Sigma(\mathtt{P}) = \bigcup_{\mathtt{P'} \in \mathcal{P}} \Sigma(\mathtt{P}, \mathtt{P'})$ of DECs, and its trust relationships. These ontologies may be pairwise inconsistent due to the DECs and the database facts. We could easily extend our framework to handle DECs that contain views, i.e. defined relational predicates. This kind of consistency issues also emerge when aligning ontologies [13]. Our notion of DEC corresponds to concept inclusion in the ontological scenario. However, the DECs we can handle can be much more general than inclusions. In our case, it has to be emphasized that, whenever possible, inconsistencies are solved at query time.

Example 1. Peers P1 and P2 have relational schemas $\mathcal{R}(\mathtt{P1}) = \{R^1, S^1\}$, $\mathcal{R}(\mathtt{P2}) = \{R^2, S^2\}$, resp. Here, P1 is connected to peer P2 by $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall xy(R^2(x, y) \wedge S^2(y, z) \rightarrow R^1(x, y, z)), \forall x(S^1(x) \rightarrow S^2(5, x))\}$, and it trusts P2 more than itself.



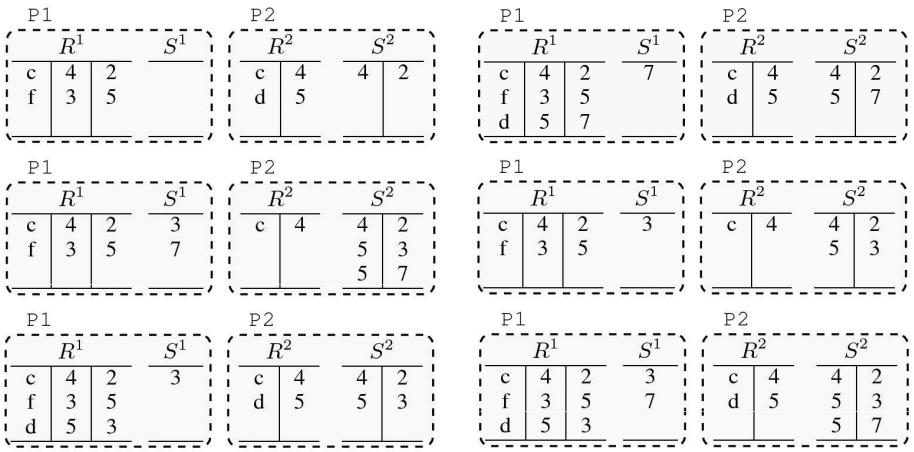
If a query is posed to P1, it has to adjust its own data so that the DECs with P2 are satisfied. To check the satisfaction, peer P1 will ask P2 for its data. Since P2 has no DECs with other peers, it will return to P1 its physical data, without any modification. Here, the data in P1 together with the data in P2 do not to satisfy the first DEC. In general, such an inconsistency could be solved by virtually removing $\langle d, 5\rangle$ from R^2 or $\langle 5, 3\rangle$ from S^2, or inserting $\langle d, 5, 3\rangle$ into R^1. But,

since P1 trusts more the data of P2, the natural choice is to add $\langle d, 5, 3\rangle$ to R^1. In the same way, the inconsistency wrt the second DEC is solved by virtually removing tuple $\langle 7\rangle$ from S^1. In this case, there is only one neighborhood solution (instance) centered around P1:



Its restriction to P1 is the solution instance for P1, and it is used to answer the queries posed to P1. Thus, the query $\mathcal{Q}_1(x) : \exists yz R^1(x, y, z)$ returns $\{\langle c\rangle, \langle f\rangle, \langle d\rangle\}$.

If we modify this example by making P1 trust P2 as much as itself, there are several possible solutions for P1, obtained by virtually modifying both peers' data. The inconsistencies wrt the first DEC would be solved by either removing $\langle d, 5\rangle$ from R^2, or $\langle 5, 3\rangle$ from S^2 or inserting $\langle d, 5, 3\rangle$ into R^1, and the inconsistencies wrt the second DEC would be solved by either removing $\langle 7\rangle$ from S^1 or inserting $\langle 5, 7\rangle$ into S^2. There are six neighborhood solutions:



In the first neighborhood solution, $\langle 5, 3\rangle$ was removed from S^2 to solve the first inconsistency. This created a new inconsistency wrt the second DEC, which was solved by removing $\langle 3\rangle$ from S^1.

The expected answers to query $\mathcal{Q}_1(x)$ would now be $\langle c\rangle, \langle f\rangle$, that are the usual answers to $\mathcal{Q}_1(x)$ shared by the six solutions for P1. □

The definition of a solution instance for P may suggest that P can physically change other peers' data, but this is not the case. Actually, the notion of solution is used as an auxiliary notion to characterize the semantically correct answers from P's point of view. We try to avoid as much as possible the generation of material solutions instances, and 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 (cf. [4] for a survey): consistent answers to a query posed to a database that may be inconsistent wrt to certain ICs are those that are invariant under all *repairs*, i.e. the minimally repaired and consistent versions of the original instance. There are mechanisms for computing consistent answers that avoid or minimize the physical generation of repairs. In this paper, we show how disjunctive logic programs with stable models semantics [12] (or answer set programs) can be used to characterize and obtain peer consistent answers.

This paper is based on [3], which considerably extends and develops the semantics first suggested in [2]. To simplify and shorten the presentation we do not consider here local ICs, DECs with existential quantifiers nor null values in database instances. We refer to [3] for these extensions and for a general treatment of the subject.

2 A Semantics for PDESs

We assume that each peer $\mathtt{P}$ owns a database instance $D(\mathtt{P})$ conforming to a schema $\mathcal{R}(\mathtt{P})$, and $\mathcal{R}(\mathtt{P}) \cap \mathcal{R}(\mathtt{Q}) = \emptyset$ for $\mathtt{P} \neq \mathtt{Q}$. The schemas determine FO languages, e.g. $\mathcal{L}(\mathtt{P})$, $\mathcal{L}(\mathtt{P}, \mathtt{Q})$. Each peer $\mathtt{P}$, has a collection of (possibly empty) sets $\Sigma(\mathtt{P}, \mathtt{Q})$ of sentences of $\mathcal{L}(\mathtt{P}, \mathtt{Q})$, which contain the DECs from $\mathtt{P}$ to peer $\mathtt{Q}$. It could be $\Sigma(\mathtt{P}, \mathtt{Q}) \neq \Sigma(\mathtt{Q}, \mathtt{P})$. $\Sigma(\mathtt{P}) := \bigcup_{\mathtt{Q}} \Sigma(\mathtt{P}, \mathtt{Q})$. There is also a relation $trust \subseteq \mathcal{P} \times \{less, same\} \times \mathcal{P}$, with exactly one triple of the form $\langle \mathtt{P}, \cdot, \mathtt{Q} \rangle$ for each non empty $\Sigma(\mathtt{P}, \mathtt{Q})$. $\mathtt{P}$ owns (or stores) those triples of the form $\langle \mathtt{P}, \cdot, \cdot \rangle$. The intended semantics of $\langle \mathtt{P}, less(same), \mathtt{Q} \rangle \in trust$ is that $\mathtt{P}$ trusts itself less than (the same as) $\mathtt{Q}$. Here, we assume $\Sigma(\mathtt{P}, \mathtt{P}) = \emptyset$ (otherwise, see [3]).

A *universal data exchange constraint* (UDEC) between peers $\mathtt{P}$, $\mathtt{Q}$ is a first-order (FO) sentence of the form:

$$\forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \longrightarrow (\bigvee_{j=1}^{m} Q_j(\bar{y}_j) \vee \varphi)), \tag{1}$$

where the R_i, Q_j are relations in $\mathcal{R}(\mathtt{P}) \cup \mathcal{R}(\mathtt{Q})$, φ is a formula containing built-in atoms[2] only, and $\bar{x}_i, \bar{y}_j \subseteq \bar{x}$.

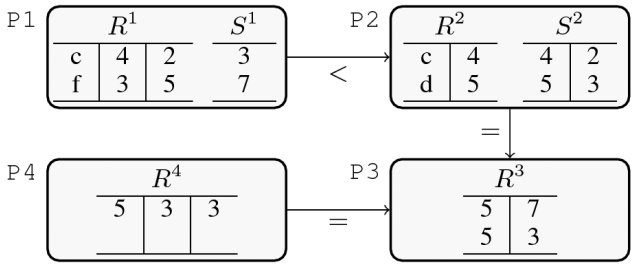
Query answering is, informally, as follows: When a peer $\mathtt{P}$ is posed a query in its local language $\mathcal{L}(\mathtt{P})$, it may have to determine, on the basis of its DECs, if its neighbors have data that is relevant to answer the query. So, it submits queries to its neighbors, whose answers may be used to answer the original query. However, before answering the query, $\mathtt{P}$ has to locally solve inconsistencies wrt to its DECs, its own data, and the data imported from the other peers. Inconsistencies are solved taking into account $\mathtt{P}$'s trust relationships. This leads to a set of virtual instances, the minimal repairs of $\mathtt{P}$'s local instance previously extended with its peers' data. In them, together with the neighbors' instances, $\mathtt{P}$'s DECs are

[2] For example, $x = 5$, $y \neq z$ and $z < 2$.

satisfied. The answers returned by P to the user are those that are true in the restrictions to $\mathcal{R}(\mathtt{P})$ of all those instances.

As expected, the solution instances for a peer will be determined not only by its relationships with its neighbors, but also by the neighbors of its neighbors, etc. An *accessibility graph* $\mathcal{G}(\mathcal{P})$ can be used to represent the connections via DECs between peers. It contains a vertex for each peer $\mathtt{P} \in \mathcal{P}$ and a directed edge from Pi to Pj if $\Sigma(\mathtt{Pi}, \mathtt{Pj}) \neq \emptyset$. An edge from Pi to Pj is labeled with "<" when $\langle \mathtt{Pi}, \mathit{less}, \mathtt{Pj} \rangle \in \mathit{trust}$, or with "=" when $\langle \mathtt{Pi}, \mathit{same}, \mathtt{Pj} \rangle \in \mathit{trust}$.[3] P' is *accessible* from P if there is a path in $\mathcal{G}(\mathcal{P})$ from P to P' or P'=P. P' is a *neighbor* of P if there is an edge from P to P'. With $\mathcal{AC}(\mathtt{P})$ and $\mathcal{N}(\mathtt{P})$ we denote the sets of peers that are accessible from P and the neighbors of P including P itself, respectively. $\mathcal{G}(\mathtt{P})$ is the restriction of $\mathcal{G}(\mathcal{P})$ to $\mathcal{AC}(\mathtt{P})$.

Example 2 (extension of example 1). The DECs are $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall xyz\ (R^2(x,y) \wedge S^2(y,z) \rightarrow R^1(x,y,z)), \forall x\ (S^1(x) \rightarrow S^2(5,x))\}$, $\Sigma(\mathtt{P2}, \mathtt{P3}) = \{\forall xy\ (S^2(x,y) \rightarrow R^3(x,y))\}$, and $\Sigma(\mathtt{P4}, \mathtt{P3}) = \{\forall xyz\ (R^3(x,y) \rightarrow R^4(x,y,3))\}$. Here, $\mathcal{N}(\mathtt{P1}) = \{\mathtt{P1}, \mathtt{P2}\}$.



If a query is posed to P1, it will send queries to P2, to check the satisfaction of the DECs in $\Sigma(\mathtt{P1}, \mathtt{P2})$. But, in order for P2 to answer those queries, it will send queries to peer P3 to check the DECs in $\Sigma(\mathtt{P2}, \mathtt{P3})$. Since P3 is not connected to any other peer, it will answer P2's queries using its material instance $D(\mathtt{P3})$. Thus, the solutions for P1 and the peer consistent answers from P1 will be affected by the peers in $\mathcal{AC}(\mathtt{P1}) = \{\mathtt{P1}, \mathtt{P2}, \mathtt{P3}\}$. Solutions for P4 will be affected by $\mathcal{AC}(\mathtt{P4}) = \{\mathtt{P4}, \mathtt{P3}\}$. □

The data distributed across different peers has to be appropriately gathered to build solution instances for a peer, and different semantics may emerge as candidates, depending on the granularity of the data sent between peers. Here we present one according to which the data that a peer P receives from a neighbor Q to build its own solutions is the *intersection of the solutions* for Q. In other terms, a peer passes *certain* data to a neighbor. After P collects this data, P uses its DECs to determine its own solutions. This is a recursive definition since the solutions for the neighbors have to be determined first, under the same semantics.

[3] In case a peer P trusts itself more than another peer, the information of the latter is irrelevant to P.

Base cases of the recursion are peers with no relevant DECs. As a consequence, this semantics requires an acyclic accessibility graph.

A database instance D of a schema $\mathcal{S}$ can be seen as a finite set of ground facts. If R is a predicate in $\mathcal{S}$, $D|\{R\}$ denotes the extension of R in D. If $\mathcal{R}(\mathtt{P}) \subseteq \mathcal{S}$, $D|\mathtt{P}$ is the restriction of D to $\mathcal{R}(\mathtt{P})$. Below, $\Delta(\cdot,\cdot)$ is used for the symmetric difference of sets.

Definition 1. Given a peer $\mathtt{P}$ and instances D, D' on schema $\bigcup_{\mathtt{Q}\in\mathcal{N}(\mathtt{P})}\mathcal{R}(\mathtt{Q})$, D' is a *neighborhood solution for* $\mathtt{P}$ *and* D if : (a) $D' \models \bigcup_{\mathtt{Q}\in\mathcal{N}(\mathtt{P})}\Sigma(\mathtt{P},\mathtt{Q})$. (b) $D'|\{R\} = D|\{R\}$ for every predicate $R \in \mathcal{R}(\mathtt{Q})$ with $(\mathtt{P}, \mathit{less}, \mathtt{Q}) \in \mathit{trust}$. (c) There is no instance D'' satisfying (a), (b), and $\Delta(D, D'') \subsetneqq \Delta(D, D')$. □

A neighborhood solution for $\mathtt{P}$ is a database for its whole neighborhood that satisfies $\mathtt{P}$'s DECs and trust relationships. A neighborhood solution stays close to the original instances (and stays the same for trustable peers): The data set that is imported or given up to satisfy the DECs is minimized. $N(\mathtt{P}, D)$ is the set of neighborhood solutions for $\mathtt{P}, D$. The set $S(\mathtt{P})$ of solution instances for $\mathtt{P}$ is recursively defined as follows:

Definition 2. For $\mathtt{P}$ with local instance $D(\mathtt{P})$, an instance D over $\mathcal{R}(\mathtt{P})$ is a *solution instance* for $\mathtt{P}$ if: (a) For $\Sigma(\mathtt{P}) = \emptyset$, $D = D(\mathtt{P})$; (b) For $\Sigma(\mathtt{P}) \neq \emptyset$, $D = \overline{D}|\mathtt{P}$, where $\overline{D} \in N(\mathtt{P}, D(\mathtt{P}) \cup \bigcup_{\mathtt{Q}\in(\mathcal{N}(\mathtt{P})\smallsetminus\{\mathtt{P}\})}\bigcap_{I\in S(\mathtt{Q})} I)$. □

Intuitively, before determining $\mathtt{P}$'s solutions, $\mathtt{P}$ has its local instance $D(\mathtt{P})$, and each neighbor $\mathtt{P'}$ has an instance for $\mathtt{P}$ that is the intersection of $\mathtt{P'}$'s solutions. This produces a combined database $\overline{D}$. Neighborhood solutions for $\mathtt{P}$ with $\overline{D}$ can be determined; and their restrictions to $\mathtt{P}$'s schema become $\mathtt{P}$'s solutions.

The peer consistent answers are the semantically correct answers to a query returned by a peer who consistently considers the data of- and trust relationships with its neighbors.

Definition 3. Let $\mathcal{Q}(\bar{x}) \in \mathcal{L}(\mathtt{P})$ be a FO query. A ground tuple $\bar{t}$ is a *peer consistent* answer (PCA) to $\mathcal{Q}$ from $\mathtt{P}$ iff $D \models \mathcal{Q}(\bar{t})$ for every $D \in S(\mathtt{P})$. □

Example 3 (example 2 continued). The solutions for $\mathtt{P1}$ require the solutions for $\mathtt{P2}$, who needs in its turn the solutions for $\mathtt{P3}$. $\mathtt{P3}$ has no DECs with other peers and its only neighborhood solution is its instance $D(\mathtt{P3})$. This is sent back to $\mathtt{P2}$, who needs to repair $\{R^2(c,4), R^2(d,5), S^2(4,2), S^2(5,3), R^3(5,7), R^3(5,3)\}$ wrt $\Sigma(\mathtt{P2},\mathtt{P3})$. As $\mathtt{P2}$ trusts $\mathtt{P3}$ the same as itself, it can modify its own data or the data it got from $\mathtt{P3}$. $\mathtt{P2}$ has two neighborhood solutions: $\{R^2(c,4), R^2(d,5), S^2(5,3), R^3(5,7), R^3(5,3)\}$ and $\{R^2(c,4), R^2(d,5), S^2(4,2), S^2(5,3), R^3(5,7), R^3(5,3), R^3(4,2)\}$, that lead to two solutions for $\mathtt{P2}$: $\{R^2(c,4), R^2(d,5), S^2(5,3)\}$ and $\{R^2(c,4), R^2(d,5), S^2(4,2), S^2(5,3)\}$.

Peer $\mathtt{P2}$ will send to $\mathtt{P1}$ their intersection: $\{R^2(c,4), R^2(d,5), S^2(5,3)\}$. Now, $\mathtt{P1}$ has to repair $\{R^1(c,4,2), R^1(f,3,5), S^1(3), S^1(7), R^2(c,4), R^2(d,5), S^2(5,3)\}$ wrt $\Sigma(\mathtt{P1},\mathtt{P2})$. Since $\mathtt{P1}$ trusts $\mathtt{P2}$ more, it will solve inconsistencies by modifying its own data, producing only one neighborhood solution: $\{R^1(c,4,2), R^1(f,3,5),$

$R^1(d,5,3)$, $S^1(3)$, $R^2(c,4)$, $R^2(d,5)$, $S^2(5,3)\}$. Thus, $S(\mathtt{P1}) = \{\{R^1(c,4,2)$, $R^1(f,3,5)$, $R^1(d,5,3)$, $S^1(3)\}\}$.

Similarly, the neighborhood solutions for P4 are $\{R^4(5,3,3), R^4(5,7,3), R^3(5,7)$, $R^3(5,3)\}$ and $\{R^4(5,3,3)$, $R^3(5,3)\}$. Thus, $S(\mathtt{P4}) = \{\ \{R^4(5,\ 3,3)$, $R^4(5,7,3)\}$, $\{R^4(5,\ 3,\ 3)\}\ \}$. Thus, if $\mathcal{Q}(x,y,z)$: $R^4(x,y,z)$ is posed to P4, its first solution instance returns $\{\langle 5,3,3\rangle, \langle 5,7,3\rangle\}$, and the second, $\{\langle 5,3,3\rangle\}$. Then, the only PCA is $\{\langle 5,3,3\rangle\}$. □

In order to answer a query, a peer may not need the whole intersection of solutions for its neighbors, but only the portions of them that are relevant to its DECs and the query at hand. This relevant and certain data can be obtained as PCAs to appropriate queries submitted to its neighbors [3].

3 Answer Set Programs and a Peer's Solutions

Solutions for a peer can be specified as the stable models of disjunctive logic programs (DLPs) [12], also called *answer set programs* [11]. These programs use annotation constants to indicate if an atom has to be virtually inserted or deleted to restore consistency:

Annotation	Atom	The tuple $P(\bar{a})$ is ...
$\mathbf{t}$	$P\_(\bar{a}, \mathbf{t})$	advised to be made true
$\mathbf{f}$	$P\_(\bar{a}, \mathbf{f})$	advised to be made false
$\mathbf{t}^\star$	$P\_(\bar{a}, \mathbf{t}^\star)$	true or becomes true
$\mathbf{t}^{\star\star}$	$P\_(\bar{a}, \mathbf{t}^{\star\star})$	true in the solution

Here, $P\_$ is a predicate obtained from the database predicate P by adding a new argument to accommodate annotations. Each peer P has a local, facts-free program that depends on its DECs. P, when posed a query, will run it with a query program, using as facts those in its local instance and the relevant ones from the intersections of the solutions of its neighbors. To get the latter, P sends to each neighbor P' queries of the form $\mathcal{Q}: R(\bar{x})$, where R is a relation of P' that appears in $\Sigma(\mathtt{P}, \mathtt{P'})$. In order to return to P the PCAs to its queries, the neighboring peers have to run their own programs. As before, they will need PCAs from their own neighbors; etc. This recursion will eventually reach peers that have no DECs, who will offer answers from their original instances to queries by other peers. Now, propagation of PCAs goes backwards until reaching P, and P gets the facts to run its program and obtain the PCAs to the original query.

Example 4 (example 3 continued). If P1 is posed the query $\mathcal{Q}_0(x) : S^1(x)$, it will need data from the intersection of the solutions of P2, to check the satisfaction of $\Sigma(\mathtt{P1}, \mathtt{P2})$. Thus, it will send to P2 the queries $\mathcal{Q}_1(x,y) : R^2(x,y)$ and $\mathcal{Q}_2(x,y) : S^2(x,y)$, expecting for PCAs to them. Now, P2 sends to P3 a single query, $\mathcal{Q}_3(x,y) : R^3(x,y)$. Since P3 has no DECs, it returns $\mathcal{Q}_3 = \{\langle 5,7\rangle, \langle 5,3\rangle\}$, directly from $D(\mathtt{P3})$. Thus, P2 has the following answer set program:

$$
\begin{array}{l}
S^2\_(x,y,\mathbf{f}) \vee R^3\_(x,y,\mathbf{t}) \leftarrow S^2\_(x,y,\mathbf{t}^\star), not R^3\_(x,y). \\
S^2\_(x,y,\mathbf{f}) \vee R^3\_(x,y,\mathbf{t}) \leftarrow S^2\_(x,y,\mathbf{t}^\star), R^3\_(x,y,\mathbf{f}). \\
\left.\begin{array}{l}
R^2\_(x,y,\mathbf{t}^\star) \leftarrow R^2\_(x,y,\mathbf{t}). \\
R^2\_(x,y,\mathbf{t}^\star) \leftarrow R^2(x,y). \\
\leftarrow R^2\_(x,y,\mathbf{t}), R^2\_(x,y,\mathbf{f}). \\
R^2\_(x,y,\mathbf{t}^{\star\star}) \leftarrow R^2\_(x,y,\mathbf{t}^\star), not R^2\_(x,y,\mathbf{f}).
\end{array}\right\} \text{Similarly for } S^2 \text{ and } R^3 \\
R^2(c,4). \quad R^2(d,5). \quad S^2(4,2). \quad S^2(5,3). \quad R^3(5,7). \quad R^3(5,3).
\end{array}
$$

The facts of this program are those in P2's instance and the PCAs from P3. The first two rules enforce the satisfaction of the DEC $\forall xy\ (S^2(x,y) \rightarrow R^3(x,y))$, e.g. the first rule specifies that if $S^2(x,y)$ is true and $R^3(x,y)$ is not in P2's database, then either $S^2(x,y)$ is deleted ($\mathbf{f}$) or $R^3(x,y)$ is inserted ($\mathbf{t}$). The other rules capture the semantics of the annotations. The fifth rule, i.e. the program denial constraint, prevents a database atom from being both inserted and deleted. Atoms annotated with $\mathbf{t}^{\star\star}$ in a stable model of the program correspond to those in the associated solution for P2. Stable models and solutions are in 1-1 correspondence. The program above is run by P2 with the queries posed by P1, e.g. for $\mathcal{Q}_1$ with query rule $Ans_1(x,y) \leftarrow R^2\_(x,y,\mathbf{t}^{\star\star})$. P2 sends to P1 the PCAs $\{\langle c,4\rangle, \langle d,5\rangle\}$ for $\mathcal{Q}_1$, and $\{\langle 5,3\rangle\}$ for $\mathcal{Q}_2$. Now, P1 has all the facts for its program:

$$
\begin{array}{l}
R^1\_(x,y,z,\mathbf{t}) \leftarrow R^2\_(x,y,\mathbf{t}^\star), S^2\_(y,z,\mathbf{t}^\star),\ not\ R^1\_(x,y,z). \\
R^1\_(x,y,z,\mathbf{t}) \leftarrow R^2\_(x,y,\mathbf{t}^\star), S^2\_(y,z,\mathbf{t}^\star), R^1\_(x,y,z,\mathbf{f}). \\
S^1\_(x,\mathbf{f}) \leftarrow S^1\_(x,\mathbf{t}^\star),\ not\ S^2\_(5,x). \\
S^1\_(x,\mathbf{f}) \leftarrow S^1\_(x,\mathbf{t}^\star), S^2\_(5,x,\mathbf{f}). \\
\left.\begin{array}{l}
R^2\_(x,y,\mathbf{t}^\star) \leftarrow R^2\_(x,y,\mathbf{t}). \\
R^2\_(x,y,\mathbf{t}^\star) \leftarrow R^2(x,y). \\
\leftarrow R^2\_(x,y,\mathbf{t}), R^2\_(x,y,\mathbf{f}). \\
R^2\_(x,y,\mathbf{t}^{\star\star}) \leftarrow R^2\_(x,y,\mathbf{t}^\star), not R^2\_(x,y,\mathbf{f}).
\end{array}\right\} \text{Similarly for } S^2 \text{ and } R^1 \\
R^1(c,4,2). \quad R^1(f,3,5). \quad R^1(d,5,3). \quad S^1(3). \quad R^2(c,4). \quad R^2(d,5). \quad S^2(5,3).
\end{array}
$$

The first two rules enforce the satisfaction of the DEC $\forall xyz\ (R^2(x,y) \wedge S^2(y,z) \rightarrow R^1(x,y,z))$; and the third and fourth rules, the second DEC $\forall x\ (S^1(x) \rightarrow S^2(5,x))$.

Now, P1 is able to peer consistently answer the original query $\mathcal{Q}_0 : S^1(x)$ by running the query rule $Ans_0(x) \leftarrow S^1\_(x,\mathbf{t}^{\star\star})$ together with the program above that specifies its solutions. The ground Ans_0-atoms in the intersection of all stable models, in this case only $Ans_0(3,\mathbf{t}^{\star\star})$, are the cautions answers from the solution program. They correspond to the PCAs, in this case only $\langle 3\rangle$. □

4 Discussion

We have illustrated the semantics on the bases of universal DECS. However, the semantics can be extended to DECs that include referential DECs. For example, we could have a peer system with the following sets of DECs:

$$\mathcal{P}_1 : \ \Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall xy\ (R^1(x,y) \rightarrow R^2(x,y)), \forall xy\ (R^2(x,y) \rightarrow R^1(x,y))\},$$
$$\Sigma(\mathtt{P2}, \mathtt{P1}) = \{\forall x(S^2(x) \rightarrow \exists y S^1(x,y))\}.$$

In this case, inconsistency wrt the latter DEC can be restored by introducing null values (into the second argument of S^1). A repair semantics based on null values for sets of ICs that include referential ICs was introduced and analyzed in [5]. This semantics can be adapted to the case of DECs in a peer data exchange system [3]. The solutions for a peer can be specified by means of answer set programs. Actually, in the case the DECs are *ref-acyclic*, i.e. without cycles through referential DECs, there is a 1-1 correspondence between solution instances and models of the program. For example, $\mathcal{P}_1$ above is ref-acyclic, whereas the following system $\mathcal{P}_2$ is not: $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall xy\ (R^1(x,y) \rightarrow \exists z\ R^2(x,z)), \forall xy\ (R^2(x,y) \rightarrow R^1(x,y))\}$. Sets of universal DECs are always ref-acyclic.

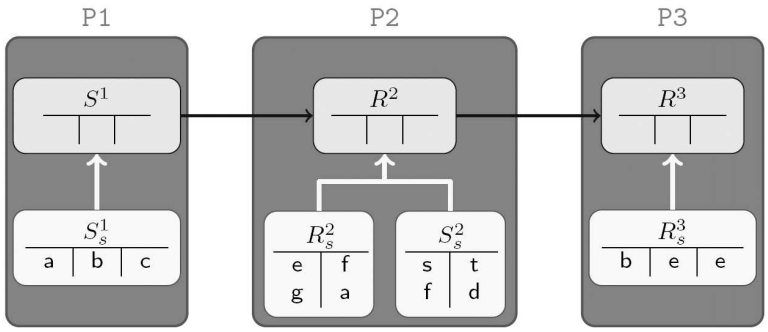
The problem of deciding peer consistent query answering is Π_2^P-complete in data [3], which matches the complexity of cautious query evaluation from DLPs. However, it is possible to identify syntactic classes of PDESs for which peer consistent query answering has a lower complexity [3]. It is also possible to identify cases in which the requirement of an acyclic accessibility graph can be relaxed [3]. This is the usual *unrestricted import case*, where the DECs are such that data is only imported into the peer (nothing is deleted), and all peers trust other peers more than themselves. In this case, peers always have solution instances, and the solution program can be replaced by a non-disjunctive program. In particular, the problem of determining if a tuple is a peer consistent answer to a query is in *coNP* [3].

It is possible to relax the conditions of ref-acyclicity and acyclicity of the peer graph, still providing a sensible semantics, and correct and complete solution-programs can be given. This is the case when, for example the cycles in the graph are not be relevant to the query. Even if the DECs are not ref-acyclic, depending on the interaction with the trust relationships, the solution program can capture exactly the set of solution instances.

Logic programs can be used to compute solutions for a peer and also PCAs. Techniques to partially compute solution instances can be useful, since we are not interested in them per se, but in the PCAs. In order to reduce the number of rules and the amount of data that are needed to run the combined query and solution program, it is possible to apply certain techniques developed for CQA. Among them we find *magic sets* for stable model semantics, and the identification of predicates that are relevant to queries and constraints [8, 9].

We can handle local ICs for peers, in addition to the DECs a peer may have with other peers. The semantics for peer solution instances can be uniformly and smoothly extended to include these local ICs. The latter will also determine the solutions for a peer, for they have to be satisfied when the local instance is virtually updated due to the presence of other peers [3]. This extension can be obtained by having "local" sets of DECs of the form $\Sigma(\mathtt{P1}, \mathtt{P1})$. For example, $\forall xyz(R^1(x,y,z) \rightarrow S^1(z)) \in \Sigma(\mathtt{P1}, \mathtt{P1}) \subseteq \mathcal{L}(\mathtt{P1})$ could be an integrity constraint on the schema $\mathcal{R}(\mathtt{P1})$ of $\mathtt{P1}$.

For comparison with related work around PDESs, we should mention [10, 7], whose semantics for PDESs have some similarities with ours (but no trust relationships). For example, in [10] DECs are of the form $cq_i \rightarrow cq_j$, where cq_i and cq_j are conjunctive queries over `Pi` and `Pj`'s schemas, resp. However, their semantics for data exchange is based on *epistemic logic*. There are no trust relationships, but implicitly, peers trust themselves less than other peers. Local ICs violations are avoided by ignoring a peer that is inconsistent wrt its local ICs. New atoms are added into a peer by interaction with other peers only if this does not produce a local IC violation.



Each peer is considered as a local virtual data integration system with *GAV local mappings* [6], in this case: $\forall xyz(S^1_s(x,y,z) \rightarrow S^1(x,y,z))$, $\forall xyz(R^2_s(x,y) \wedge R^2_s(y,z) \rightarrow R^2(x,y,z))$, $\forall xyz(R^3_s(x,y,z) \rightarrow R^3(x,y,z))$. The predicates of the form P_s correspond to local sources, and those of the form P^i correspond to the mediated schema provided by peer `Pi`. The DECs establish mapping between the latter. In this case: $\forall xy(R^2(x,y,z) \rightarrow \exists w R^1(x,y,w))$, $\forall xy(R^3(x,y,y) \rightarrow \exists uv R^3(u,x,v))$.

For query answering, the following epistemic theory is used:

$$\left.\begin{array}{l}\mathbf{K_1}(\forall xyz(S^1_s(x,y,z) \rightarrow S^1(x,y,z))) \\ \forall xy(\mathbf{K_2}(R^2(x,y,z)) \rightarrow \mathbf{K_1}(\exists w R^1(x,y,w)))\end{array}\right\} \text{Specification of } \texttt{P1}$$

$$\left.\begin{array}{l}\mathbf{K_2}(\forall xyz(R^2_s(x,y) \wedge R^2_s(y,z) \rightarrow R^2(x,y,z))) \\ \forall xy(\mathbf{K_3}(R^3(x,y,y)) \rightarrow \mathbf{K_2}(\exists uv R^3(u,x,v)))\}\end{array}\right\} \text{Specification of } \texttt{P2}$$

$$\left.\begin{array}{l}\mathbf{K_3}(\forall xyz(R^3_s(x,y,z) \rightarrow R^3(x,y,z)))\end{array}\right\} \text{Specification of } \texttt{P3}$$

$\mathbf{K_i}\phi$ is interpreted as ϕ is known by peer `Pi`. The idea behind using the epistemic theory is that data that is known (or certain) is passed from local sources to mediated schemas and from peers to other peers. A tuple $\bar{t}$ is a peer consistent answer to a query $\mathcal{Q}$ posed to peer `Pi` if $\mathbf{K_i}\mathcal{Q}(\bar{t})$ is a logical consequence of the epistemic theory.

An advantage of this approach is that the semantics can be applied in the presence of cycles. However, possibly the whole epistemic theory has to be used by a peer `Pi` to do query answering, which requires not only data, but also the

mappings and DECs; and this not only of its neighbors, but of all accessible peers.

Our approach can be easily and uniformly adapted in order to make each peer a local data integration system. For this, the specifications and answer set programs for virtual data integration introduced in [1] can be used in combination with those presented here.

We emphasize that the DECs we can handle are more general that those found in related work, including mappings between ontologies, which -when the latter are merged- requires addressing the inconsistencies that naturally emerge [13]. In particular, our DECs may have relations of both peers on the two sides of the implication. In [2] a fully developed example of this kind can be found.

Our semantics allows for inconsistent peers and inconsistencies between peers, without unraveling logical reasoning or having to exclude peers whose data participates in inconsistencies. In this sense, we may say that our semantics is *inconsistency tolerant*. Actually, it is even more than this: inconsistency is the driving and guiding force behind the process of data exchange.

Acknowledgements. Research supported by NSERC and a CITO/IBM-CAS Student Internship. L. Bertossi is Faculty Fellow of IBM CAS (Toronto Lab.). Part of this research was done when L. Bertossi visited the University of Edinburgh in 2007. The invitation and hospitality are much appreciated.

References

[1] Bertossi, L., Bravo, L.: Consistent Query Answers in Virtual Data Integration Systems. In: Bertossi, L., Hunter, A., Schaub, T. (eds.) Inconsistency Tolerance. LNCS, vol. 3300, pp. 42–83. Springer, Heidelberg (2004)

[2] Bertossi, L., Bravo, L.: Query Answering in Peer-to-Peer Data Exchange Systems. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 476–485. Springer, Heidelberg (2004)

[3] Bertossi, L., Bravo, L.: The Semantics of Consistency and Trust in Peer Data Exchange Systems. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 107–122. Springer, Heidelberg (2007)

[4] Bertossi, L.: Consistent Query Answering in Databases. ACM Sigmod Record 2(35), 68–76 (2006)

[5] Bravo, L., Bertossi, L.: Semantically Correct Query Answers in the Presence of Null Values. In: Grust, T., Höpfner, H., Illarramendi, A., Jablonski, S., Mesiti, M., Müller, S., Patranjan, P.-L., Sattler, K.-U., Spiliopoulou, M., Wijsen, J. (eds.) EDBT 2006. LNCS, vol. 4254, pp. 336–357. Springer, Heidelberg (2006)

[6] Lenzerini, M.: Data Integration: A Theoretical Perspective. In: Proc. Symposium on Principles of Database Systems (PODS 2002), pp. 233–246. ACM Press, New York (2002)

[7] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Inconsistency Tolerance in P2P Data Integration: An Epistemic Logic Approach. In: Bierman, G., Koch, C. (eds.) DBPL 2005. LNCS, vol. 3774, pp. 90–105. Springer, Heidelberg (2005)

[8] Caniupan, M., Bertossi, L.: The Consistency Extractor System: Querying Inconsistent Databases using Answer Set Programs. In: Prade, H., Subrahmanian, V.S. (eds.) SUM 2007. LNCS (LNAI), vol. 4772, pp. 74–88. Springer, Heidelberg (2007)
[9] Eiter, T., Fink, M., Greco, G., Lembo, D.: Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In: Palamidessi, C. (ed.) ICLP 2003. LNCS, vol. 2916, pp. 163–177. Springer, Heidelberg (2003)
[10] Franconi, E., Kuper, G., Lopatenko, A., Zaihrayeu, I.: A Distributed Algorithm for Robust Data Sharing and Updates in P2P Database Networks. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 446–455. Springer, Heidelberg (2004)
[11] Gelfond, M., Leone, N.: Logic Programming and Knowledge Representation - The A-Prolog Perspective. Artificial Intelligence 138(1-2), 3–38 (2002)
[12] Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9(3/4), 365–386 (1991)
[13] Serafini, L., Borgida, A., Tamilin, A.: Aspects of Distributed and Modular Ontology Reasoning. In: Proc. International Joint Conference on Artificial Intelligence (IJCAI 2005), pp. 570–575. Morgan Kaufmann, San Francisco (2005)