The Semantics of Consistency and Trust in Peer Data Exchange Systems

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 propose and investigate a semantics for *peer data exchange systems* (or peer data management systems) where different peers are pairwise related to each other by means of data exchange constraints and trust relationships. These two elements plus the data at the peers' sites and the local integrity constraints for a peer are made compatible via the proposed semantics by determining a set of *solution instances*, which are the intended virtual instances for the peer. The semantically correct answers from a peer to a query, called its *peer consistent answers*, are defined as those answers that are invariant under all its different solution instances. We show that solution instances can be specified as the models of logic programs with a stable model semantics.

1 Introduction

A peer data exchange system (PDES) consists of a finite set of peers $\{\mathtt{P}_1, \ldots \mathtt{P}_n\}$, each of them with a local database instance. Peers may be pairwise related by means of logical sentences, called *data exchange constraints* (DECs), which are expressed in terms of the participating schemas and are expected to be satisfied by the combined data. Furthermore, a peer P may trust its data the same as or less than other peers' data, i.e. there may be trust relationships between pairs of peers. We may also have integrity constraints (ICs) that are local to each peer.

The DECs could be seen as ICs on a global database obtained by conceptually putting together all the peers' schemas and data. Most likely, these DECs will not be satisfied in this global database, but virtually enforcing their satisfaction at query time has the effect of shipping (sub)queries and data between peers. Actually, in such a PDES, a query $\mathcal{Q}$ is posed to a peer P, who, in order to answer it, may need to consider both its own data and the data stored at other peers' sites that are related to P by DECs. Keeping P's DECs satisfied at query time may imply getting data from other peers to complement P's data, but also not using part of its own data.

The decision by a peer P on what other data to consider does not depend only on its DECs, but also on the *trust relationships* that P has with other peers. For example, if peer P trusts peer Q's data more than its own, P will accommodate its data to Q's in order to keep the DECs between them satisfied.

In order for P to return meaningful answers, its local semantic constraints have to be taken into account. The consistency of its instance should be preserved when it is virtually updated due to the interaction with other peers. In consequence, still at query time, P may also need to "virtually repair" its data.

Our semantics makes all these elements compatible, by defining a set of virtual global instances called *solution instances* (or simply solutions) for a peer. In consequence, the "data" for a peer, the one in its solution instances, depends upon its instance, the local instances of the related peers, the satisfaction of the DECs, and the satisfaction by P of its local ICs. After that, having a precise definition of the intended solution instances for a peer P, the *peer consistent answers* (PCAs) from peer P to query $\mathcal{Q}$ are defined as those answers that can be retrieved from *every* possible solution for P.

The definition of solution for P may suggest that P may physically change other peers' data, 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, where consistent answers to a query posed to a database, which is possibly inconsistent wrt to a given set of ICs, are defined as those answers that can be retrieved from every minimally repaired version of the original instance. Methods have been developed for computing these answers without having to physically repair the given instance [1, 8, 9].

Our work goes in the direction of semantic approaches to peer-to-peer data exchange [26, 28, 27, 13, 20, 7, 14, 22]. In [7] trust relationships were introduced for the first time in this scenario and the notions of solution instance for a peer and of peer consistent answer to a query were introduced. The case of a peer and its immediate neighbors was considered and investigated. However, the situation where a peer is related by logical transitivity via DECs to other, non neighboring peers was not modelled. Actually, in [7] it was only indicated that giving a semantics to solution instances for a peer that consider the transitive relationships to other peers could be done by using logic programs with stable model semantics. We fully develop this idea here. First, we provide a general model-theoretic definition of solutions for a peer, including the transitive case, and next, we specify the solutions as the models of disjunctive logic programs with stable model semantics [23], aka. *answer set programs* [24, 3].

Logic programs can capture the different ways the system stabilizes after satisfying the DECs, the trust relationships, and the local ICs. Disjunctive programs allow for the specification of alternative virtual updates on data sources under certain conditions. We propose appropriate logic programs and then we establish that there is a one-to-one correspondence between the set of solution instances for a peer and the set of stable models of the program. Logic programs provide an expressive language for representing and specifying the alternative solutions for a peer, and become executable specifications for computing peer consistent answers. This approach has been exploited in CQA, where database repairs are specified as stable models of a program [2, 25, 4, 16].

2 A Semantics for PDESs

We consider peers that have mutually disjoint relational schemas; but all of them share a possibly infinite database domain $\mathcal{U}$. Peers are denoted by A, B, P, Q, ...

Definition 1. A *peer data exchange system* $\mathfrak{P}$ consists of: (a) A finite set $\mathcal{P}$ of peers, with each peer P owning a relational database schema $\mathcal{R}(\mathtt{P})$, and

a database instance $D(\mathtt{P})$ conforming to schema $\mathcal{R}(\mathtt{P})$. The schemas determine FO languages, e.g. $\mathcal{L}(\mathtt{P})$, $\mathcal{L}(\mathtt{P},\mathtt{Q})$. (b) For each peer $\mathtt{P}$, collections $\Sigma(\mathtt{P},\mathtt{Q})$ of sentences of $\mathcal{L}(\mathtt{P},\mathtt{Q})$, which contain the DECs between $\mathtt{P}$ and a peer $\mathtt{Q}$. Here, $\Sigma(\mathtt{P}) := \bigcup_{\mathtt{Q}} \Sigma(\mathtt{P},\mathtt{Q})$ and $\Sigma := \bigcup_{\mathtt{P}\in\mathcal{P}} \Sigma(\mathtt{P})$. (c) For each peer $\mathtt{P}$, a set of $\mathcal{L}(\mathtt{P})$-sentences $IC(\mathtt{P})$ that are ICs on $\mathcal{R}(\mathtt{P})$. Here, $IC = \bigcup_{\mathtt{P}\in\mathcal{P}} IC(\mathtt{P})$. (d) 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 $\Sigma(\mathtt{P},\mathtt{Q})$. □

The intended semantics of $(\mathtt{A}, less, \mathtt{B}) \in trust$ is that 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 $\mathtt{B}$.

Definition 2. (a) A *universal data exchange constraint* (UDEC) between peers $\mathtt{P1}$, $\mathtt{P2}$ is a first-order (FO) formula of 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 $\mathcal{R}(\mathtt{P1}) \cup \mathcal{R}(\mathtt{P2})$, φ is a formula containing built-in atoms[1] only, and $\bar{x}_i, \bar{y}_j \subseteq \bar{x}$. (b) A *referential data exchange constraint* (RDEC) between peers $\mathtt{P1}$, $\mathtt{P2}$ is an $\mathcal{L}(\mathtt{P1},\mathtt{P2})$-sentence of the form:

$$\forall \bar{x}(R(\bar{x}) \longrightarrow \exists \bar{y}\ Q(\bar{x}', \bar{y})); \tag{2}$$

where $R, Q \in \mathcal{R}(\mathtt{P1}) \cup \mathcal{R}(\mathtt{P2})$, and $\bar{x}' \subseteq \bar{x}$. □

Notice that the sets of DECs $\Sigma(\mathtt{P1},\mathtt{P2})$ and $\Sigma(\mathtt{P2},\mathtt{P1})$ can be different. When exchanging or repairing data, the existential quantifier in RDECs will be interpreted as a null value. By having one database atom in the consequent, we avoid existential joins that, when filled with nulls, do not have a clear semantics (cf. [11] for a discussion and a FO semantics of nulls in SQL databases).[2]

Example 1. Consider a PDES $\mathfrak{P}$ with four peers and $\mathcal{R}(\mathtt{P1}) = \{R^1(\cdot,\cdot)\}$, $\mathcal{R}(\mathtt{P2}) = \{R^2(\cdot,\cdot), S^2(\cdot,\cdot)\}$, $\mathcal{R}(\mathtt{P3}) = \{R^3(\cdot,\cdot)\}$, $\mathcal{R}(\mathtt{P4}) = \{R^4(\cdot,\cdot,\cdot)\}$, and DECs:

$\Sigma(\mathtt{P1},\mathtt{P2}) = \{\forall xy(R^2(x,y) \to R^1(x,y))\}$,
$\Sigma(\mathtt{P2},\mathtt{P3}) = \{\forall xy(R^2(x,y) \wedge R^3(x,y) \to \mathbf{false})\}$,
$\Sigma(\mathtt{P4},\mathtt{P2}) = \{\forall xyz(R^2(x,y) \wedge S^2(y,z) \to R^4(x,y,z))\}$,
$\Sigma(\mathtt{P4},\mathtt{P3}) = \{\forall xy(R^3(x,y) \to \exists z R^4(x,y,z))\}$.

Here **false** in the second DEC is a built-in atom that is false in every instance, so the DEC specifies that relations R^2 and R^3 are disjoint. The DECs in $\Sigma(\mathtt{P1},\mathtt{P2})$, $\Sigma(\mathtt{P2},\mathtt{P3})$ and $\Sigma(\mathtt{P4},\mathtt{P2})$ are UDECs and the one in $\Sigma(\mathtt{P4},\mathtt{P3})$ is a RDEC. Finally, we could have $trust = \{(\mathtt{P1}, less, \mathtt{P2}), (\mathtt{P2}, same, \mathtt{P3}), (\mathtt{P4}, less, \mathtt{P2}), (\mathtt{P4}, less, \mathtt{P3})\}$. Peer $\mathtt{P1}$ trusts $\mathtt{P2}$ more than itself, and it will import all the data from R^2 into its table R^1. On the other hand, peer $\mathtt{P2}$ trusts peer $\mathtt{P3}$ as much as itself, and its DEC states that it is not possible to have the same tuple in both R^3 and R^2. □

Local ICs $IC(\mathtt{P})$ are also of the form in Definition 2, but with all database predicates in $\mathcal{R}(\mathtt{P})$. So, we can identify $IC(\mathtt{P})$ with $\Sigma(\mathtt{P},\mathtt{P})$. All the common ICs found in database practice can be accommodated into these syntactic classes.

[1] For example, $x \neq 3$, $y = z$ and $z > 3$.

[2] Our framework can be easily adapted to cases where, instead of *null*, one uses arbitrary elements of the database domain [12] or labelled nulls [29].

In particular, using the atom **false**, denial constraints are of the form (1). In Example 1, we could have $IC(\mathtt{P2}) = \{\forall x \forall y(R^2(x,y) \wedge S^2(x,y) \rightarrow \mathbf{false})\}$.

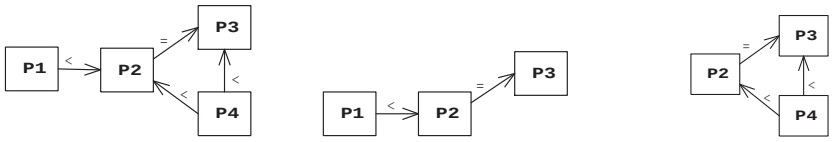
Each peer P is responsible for maintaining its material instance consistent with respect to its ICs $IC(\mathtt{P})$, independently from other peers; and we may assume that this is the case. However, our semantics works the same if we allow peers with locally inconsistent data. According to our semantics presented below, unsatisfied local ICs are considered when solution instances are specified, which is done by using a repair semantics and techniques developed for CQA [9, 6].

When a peer P is posed a query, it may have to submit queries to other peers according to its DECs, using data in other peers' relations appearing in them. Data brought from other peers will possibly cause virtual updates on P's data, which may create virtual violations of P's local ICs, which has to be considered.

From the perspective of a peer P, its own database may be inconsistent with respect to the data owned by another peer Q it trusts more or the same, and the DECs in $\Sigma(\mathtt{P},\mathtt{Q})$. When P queries its own database, the answers from P should be consistent with $\Sigma(\mathtt{P},\mathtt{Q})$ and its own ICs $IC(\mathtt{P})$. In principle, P, which is not allowed to change other peers' data, could try to physically repair its database in order to satisfy $\Sigma(\mathtt{P}) \cup IC(\mathtt{P})$. This is not a realistic approach. Rather, P should solve its semantic conflicts at query time. This leads to a set of virtual instances, the minimal repairs of P's local database, where P's DECs and *IC*s are satisfied, while respecting P's trust relationships to other peers. The answers returned by P to the user are those that are true of all these instances.

The solution instances of a peer will be determined not only by its relationships with its neighbors, but also by the neighbors of its neighbors, etc.

Definition 3. (a) The *accessibility graph* $\mathcal{G}(\mathfrak{P})$ of a PDES $\mathfrak{P}$ contains a vertex for each peer $\mathtt{P} \in \mathcal{P}$ and a directed edge from Pi to Pj if $\Sigma(\mathtt{Pi},\mathtt{Pj})$ is non empty. An edge from Pi to Pj is labelled with "<" when $(\mathtt{Pi}, \mathit{less}, \mathtt{Pj}) \in \mathit{trust}$, and with "=" when $(\mathtt{Pi}, \mathit{same}, \mathtt{Pj}) \in \mathit{trust}$.[3] (b) Peer P' is *accessible* from P if there is a path in $\mathcal{G}(\mathfrak{P})$ from P to P' or if P'=P. Peer P' is a *neighbor* of P if there is an edge from P to P' in $\mathcal{G}(\mathfrak{P})$, or if P' = 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, respectively. For $\mathtt{P} \in \mathcal{P}$, $\mathcal{G}(\mathtt{P})$ is the restriction of $\mathcal{G}(\mathfrak{P})$ to $\mathcal{AC}(\mathtt{P})$. □



(a) $\mathcal{G}(\mathfrak{P})$ (b) $\mathcal{G}(\mathtt{P1})$ (c) $\mathcal{G}(\mathtt{P4})$

Fig. 1. Graphs for Example 2

Example 2. (Example 1 continued) In this system, $\mathcal{AC}(\mathtt{P1}) = \{\mathtt{P1},\mathtt{P2},\mathtt{P3}\}$, $\mathcal{AC}(\mathtt{P2}) = \{\mathtt{P2},\mathtt{P3}\}$, $\mathcal{AC}(\mathtt{P3}) = \{\mathtt{P3}\}$, $\mathcal{AC}(\mathtt{P4}) = \{\mathtt{P2},\mathtt{P3},\mathtt{P4}\}$, $\mathcal{N}(\mathtt{P1}) = \{\mathtt{P1},\mathtt{P2}\}$, $\mathcal{N}(\mathtt{P2}) = \{\mathtt{P2},\mathtt{P3}\}$, $\mathcal{N}(\mathtt{P3}) = \{\mathtt{P3}\}$, and $\mathcal{N}(\mathtt{P4}) = \{\mathtt{P2},\mathtt{P3},\mathtt{P4}\}$. □

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

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. We develop one of them,[4] 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. After P collects this data, only P's DECs and ICs are considered. This is a recursive definition since the solutions for the neighbors have to be determined, under the same semantics. Base cases of the recursion are peers with no relevant DECs. In consequence, this semantics requires an acyclic accessibility graph.

In [26] problematic cases involving cyclic dependencies through DECs are identified, which implicitly involve a cyclic accessibility graph. For example, we may have a PDES $\mathfrak{P}$ with $\mathcal{P} = \{\mathtt{P1}, \mathtt{P2}, \mathtt{P3}\}$, with relations $R^1(\cdot), R^2(\cdot), R^3(\cdot)$, resp., and DECs $\Sigma(\mathtt{P1}) = \{\forall x(R^2(x) \rightarrow R^1(x))\}$, $\Sigma(\mathtt{P2}) = \{\forall x(R^3(x) \rightarrow R^2(x))\}$, $\Sigma(\mathtt{P3}) = \{\forall x(R^1(x) \rightarrow R^3(x))\}$, each of them satisfied only by importing data into the peer who owns the DEC. The implicit trust relation $\{(\mathtt{P1}, \mathit{less}, \mathtt{P2}), (\mathtt{P2}, \mathit{less}, \mathtt{P3}), (\mathtt{P3}, \mathit{less}, \mathtt{P1})\}$ makes $\mathcal{AC}(\mathfrak{P})$ cyclic. In [26] it is assumed that no cycles of this kind appear. In the following, we will also assume that $\mathcal{G}(\mathfrak{P})$ is acyclic, and then for each particular peer P, $\mathcal{G}(\mathtt{P})$ is acyclic.

Null values will be used to satisfy referential DECs and local referential ICs; and the repair semantics based on introduction of null values, presented and developed in [11, 10] for single relational databases and RICs, can be adapted here, but now taking into account the trust relationships. Data sources at the peers' sites may contain null values that, as those used to satisfy referential constraints, will have a semantics that corresponds to the way nulls are handled by DBMSs that follow the SQL standard. In particular, there is only one constant, *null*, that is used as the null value.[5] We use a semantics for IC satisfaction in the presence of nulls that generalizes the one implemented in DBMSs, and coincides with the first-order notion of formula satisfaction in databases without nulls [10].

A formal development of the notion of constraint satisfaction, denoted $D \models \psi$, with D possibly containing *null* and ψ a constraint, can be found in [11, 10] (see Appendix B for a review). What matters most for the rest of this paper is that the satisfaction of a constraint ψ depends upon the presence of *null* in attributes of database relations which are also *relevant attributes* for ψ.

Solutions for a peer should stay close to its original physical instance while satisfying the DECs and local ICs. We do not want to import or give up more data than strictly required to satisfy the constraints. To formalize this idea, we first need to compare tuples that may contain *null*. A constant c provides *less or equal information* than a constant d, denoted $c \sqsubseteq d$, iff c is *null* or $c = d$ [30]. A tuple $\bar{t}_1 = (c_1, \ldots, c_n)$ provides less or equal information than $\bar{t}_2 = (d_1, \ldots, d_n)$, denoted $\bar{t}_1 \sqsubseteq \bar{t}_2$, iff $c_i \sqsubseteq d_i$ for every $i = 1, \ldots, n$. Finally, $\bar{t}_1 \sqsubset \bar{t}_2$ means $\bar{t}_1 \sqsubseteq \bar{t}_2$

[4] In [10] also two other alternative semantics are fully developed and compared, in particular establishing some conditions under which they coincide or differ. The other semantics assume that more detailed information, such as mappings and trust relationships, can be sent between peers.

[5] This null is obviously different from the multiple labelled null values that are considered in data exchange for satisfying existential quantifiers (cf. [29] for a survey).

and $\bar{t}_1 \neq \bar{t}_2$. In the following, a database instance is identified with a finite set of ground database atoms; and $\Delta(\cdot,\cdot)$ denotes the symmetric difference of sets.

Definition 4. Let D, D', D'' be database instances for the same schema. It holds that $D' \leq_D D''$ iff for every $P(\bar{a}) \in \Delta(D, D')$, there exists $P(\bar{a}')$, such that: (a) $P(\bar{a}') \in \Delta(D, D'')$; (b) $\bar{a} \sqsubseteq \bar{a}'$; and (c) if $\bar{a} \sqsubset \bar{a}'$, then $P(\bar{a}') \notin \Delta(D, D')$. Finally, $D'' <_D D'$ means $D'' \leq_D D'$ but not $D' \leq_D D''$. □

If $D' \leq_D D''$, we say that D' is closer to D than D''. Condition (c) in Def. 4 ensures that a database that adds to D a tuple with *null* is closer to D than other that adds other constant. This condition will later ensure that the satisfaction of RDECs is enforced by using *null*.

For an instance D of a schema $\mathcal{S}$, and $\mathcal{S}'$ a subschema of $\mathcal{S}$, $D|\mathcal{S}'$ denotes the restriction of D to $\mathcal{S}'$. Thus, if R is a predicate in $\mathcal{S}$ and D is an instance for $\mathcal{S}$, $D|\{R\}$ denotes the extension of R in D. If $\mathcal{R}(\mathtt{P}) \subseteq \mathcal{S}$, $D|\mathtt{P}$ is the restriction of D to $\mathcal{R}(\mathtt{P})$. A neighborhood solution for $\mathtt{P}$ and a database for its whole neighborhood is a closest database that satisfies $\mathtt{P}$'s DECs, ICs, and trust relationships.

Definition 5. Given a peer $\mathtt{P}$ in a PDES $\mathfrak{P}$ 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) $D' \models \bigcup_{\mathtt{Q}\in\mathcal{N}(\mathtt{P})} \Sigma(\mathtt{P},\mathtt{Q}) \cup IC(\mathtt{P})$. (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'' that satisfies (a) and (b), and such that $D'' <_D D'$. □

We do not require in (a) $IC(\mathtt{Q})$ to be satisfied, because $\mathtt{Q}$ will move data to $\mathtt{P}$'s site, where inconsistencies will be solved locally, according to Definition 6, where $S(\mathtt{P})$ denotes the set of solutions for peer $\mathtt{P}$.

Definition 6. Given a peer $\mathtt{P}$ in a PDES $\mathfrak{P}$ with local instance $D(\mathtt{P})$, an instance D over $\mathcal{R}(\mathtt{P})$ is a *solution instance* for $\mathtt{P}$ if: (a) $D = D(\mathtt{P})$ and $\Sigma(\mathtt{P}) = \emptyset$; or (b) $\Sigma(\mathtt{P}) \neq \emptyset$, $D = \overline{D}|\mathtt{P}$ where $\overline{D}$ is a neighborhood solution for $\mathtt{P}$ and the database instance $D(\mathtt{P}) \cup \bigcup_{\mathtt{Q}\in(\mathcal{N}(\mathtt{P})\setminus\{\mathtt{P}\})} \bigcap_{I\in S(\mathtt{Q})} I$ over schema $\bigcup_{\mathtt{Q}\in\mathcal{N}(\mathtt{P})} \mathcal{R}(\mathtt{Q})$. □

Intuitively, before constructing $\mathtt{P}$'s solutions, $\mathtt{P}$ has its local instance $D(\mathtt{P})$ and each of its neighbors has as local instance the intersection of its own solutions. This produces a combined database. After that, the solutions for $\mathtt{P}$ are obtained by restricting to $\mathtt{P}$ the neighborhood solutions for the combined instance. The neighborhood solution captures the minimal virtual updates that are necessary to satisfy the DECS and local ICs. As there may be several neighborhood solutions, several solutions for a peer are possible.

Example 3. (Example 1 and 2 continued) Consider the following instances of peers $\mathtt{P1}$, $\mathtt{P2}$ and $\mathtt{P3}$: $D(\mathtt{P1}) = \{R^1(a,2)\}$, $D(\mathtt{P2}) = \{R^2(c,4), R^2(d,5)\}$, $D(\mathtt{P3}) = \{R^3(c,4)\}$ and $D(\mathtt{P4}) = \{R^4(d,5,1)\}$. If we want the solutions for $\mathtt{P1}$, the solutions for $\mathtt{P2}$ are needed, who will need in turn the solutions for $\mathtt{P1}$. Since $\mathtt{P3}$ has no DECs with other peers, its only neighborhood solution is it local instance $D(\mathtt{P3})$. This data is sent back to $\mathtt{P2}$, who needs to repair $\{R^2(c,4), R^2(d,5), R^3(c,4)\}$ now 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 $\mathtt{P3}$. There are two neighborhood solutions for $\mathtt{P2}$: $\{R^2(c,4), R^2(d,5)\}$ and $\{R^2(d,5), R^3(c,4)\}$, that lead to two solutions for $\mathtt{P2}$: $\{R^2(c,4), R^2(d,5)\}$ and $\{R^2(d,5)\}$. Peer $\mathtt{P2}$ will send to $\mathtt{P1}$ the intersection of its solutions: $\{R^2(d,5)\}$. Now, $\mathtt{P1}$ has to repair $\{R^1(a,2), R^2(d,5)\}$ wrt

$\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall xy\ (R^2(x,y) \rightarrow R^1(x,y))\}$. Since P1 trusts its own data less than the data of P2, it will solve inconsistencies by modifying its own data. There is only one neighborhood solution, $\{R^1(a,2), R^2(d,5), R^1(d,5)\}$, and the solution for P1 is $\{R^1(a,2), R^1(d,5)\}$.

To compute the solutions for P4, the solutions of P2 and P3 are computed as shown before. Neighborhood solutions for P4 are obtained by repairing $\{R^4(d,5,1), R^2(d,5), R^3(c,4)\}$ wrt $\Sigma(\mathtt{P4}, \mathtt{P2})$, and $\Sigma(\mathtt{P4}, \mathtt{P3})$. The DECs in $\Sigma(\mathtt{P4}, \mathtt{P2})$ are already satisfied, but not the ones in $\Sigma(\mathtt{P4}, \mathtt{P3})$. Since P4 trusts the data in P3 more, a repair is obtained by adding a tuple with *null* into P4. The unique neighborhood solution for P4 is $\{R^4(d,5,1), R^2(d,5), R^3(c,4),\ R^4(c,4,null)\}$. Consequently, $S(\mathtt{P4}) = \{\{R^4(d,5,1),\ R^4(c,4,null)\}\}$. □

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 7. Given a FO query $\mathcal{Q}(\bar{x}) \in \mathcal{L}(\mathtt{P})$ posed to P, a ground tuple $\bar{t}$ is a *peer consistent* answer (PCA) to $\mathcal{Q}$ from P iff $D \models \mathcal{Q}(\bar{t})$ for every solution instance D for P. □

Example 4. (Example 3 continued) If P2 is posed the query $\mathcal{Q}$: $R^2(x,y)$, from its first solution instance we get $\{(c,4),(d,5)\}$, and from the second, $\{(d,5)\}$. Therefore, the only PCA from P2 is $\{(d,5)\}$. □

Even in the absence of cycles in $\mathcal{G}(\mathfrak{P})$, there may be no solutions for a peer (cf. Example 10 in Appendix A). Furthermore, still with acyclic $\mathcal{G}(\mathfrak{P})$, *the decision problem of peer consistent answering*, i.e. deciding if a tuple is a PCA to a query, may be undecidable if consistency wrt RDECs is achieved using arbitrary values in the domain.[6] However, using *null* instead avoids this problem, making the problem decidable. Actually, by reduction from CQA to PCA and known results on the *data complexity* of CQA [11], we obtain

Theorem 1. The problem of peer consistent answering is Π_2^P-complete. □

3 Answer Set Programs and the Solutions for a Peer

In order to define the solutions for a peer P, we have to consider P's relevant peers, which are those in $\mathcal{AC}(\mathtt{P})$. The presence of cycles, through trust relationships or constraints (DECs or ICs), have an impact on the semantics. The former cycles appear in a cyclic $\mathcal{G}(\mathtt{P})$. The latter appear when the DECs and local ICs of peers in $\mathcal{AC}(\mathtt{P})$ put together present cycles through the implications that involve an RDEC or a local referential IC (a RIC). Sets of local ICs of this kind are called *RIC-cyclic* in [11]. For example, $IC = \{\forall x(S(x) \rightarrow Q(x)),\ \forall x\ (Q(x) \rightarrow S(x)),\ \forall x(Q(x) \rightarrow \exists y\ T(x,y))\}$ is not RIC-cyclic, whereas $IC' = IC \cup \{\forall xy\ (T(x,y) \rightarrow Q(y))\}$ is, because there is a cycle involving the RIC $\forall x(Q(x) \rightarrow \exists y\ T(x,y))$. RIC-cyclicity at the level of local ICs may lead to more solutions than intended when capturing the repair semantics by means of logic programs [11].

[6] The undecidability result for CQA in [12] can be reconstructed in our framework, because even with $\mathcal{G}(\mathfrak{P})$ acyclic, DECs can have ref-cycles (cf. Example 5).

In order to deal with the new issues arising in PDESs, we will assume that, for each peer P, $IC(\mathtt{P})$ is RIC-acyclic (cf. Section 4 for a discussion). Cycles through DECs and ICs will be crucial for a logic programming-based specification of solutions for a peer. We will say that the PDES $\mathfrak{P}$ is *ref-acyclic* when in $\Sigma \cup IC$ there are no cycles that involve an RDEC or a RIC.

As Example 5 below shows, even assuming the acyclicity of $\mathcal{G}(\mathfrak{P})$, and RIC-acyclicity at the level of local ICs (or no local ICs at all), we may have ref-cycles in the set of all DECs. This is due to the generality of DECs, where we can have relations of any of the two peers on both sides of the implication.

Example 5. Peers P1, P2 have relations R^1, R^2, resp. $\Sigma(\mathtt{P1}) = \{\forall xz(R^1(x,z) \rightarrow \exists y R^2(x,y)), \forall xz(R^2(x,z) \rightarrow \exists y R^1(x,y)\}$, $\Sigma(\mathtt{P2}) = \emptyset$; and $(\mathtt{P1}, \mathit{less}, \mathtt{P2}) \in \mathit{trust}$. Here, $\mathcal{AC}(\mathfrak{P})$ is acyclic, but $\Sigma(\mathtt{P1}) \cup \Sigma(\mathtt{P2})$ has a ref-cycle. □

Now we will show how to specify solutions for a peer, given instances for the other peers, as the stable models of disjunctive logic programs. These programs use annotation constants to indicate the atoms that may become true or false (virtually inserted or deleted) in order to satisfy the DECs and local ICs. For each database predicate P we generate a new copy $P\_$ with an extra argument to accommodate the annotation. In $P\_(\bar{a}, \mathbf{t_a})$, annotation $\mathbf{t_a}$ means that the atom is advised to be made true; and $\mathbf{f_a}$, that the atom should be made false. For each DEC and local IC ψ, a rule captures through its disjunctive head the alternative virtual updates that can be performed to satisfy ψ (cf. rules 2. and 3. in Definition 9 for DECs, and 4. and 5. for local ICs).

Annotation $\mathbf{t}^\star$ indicates that the atom is true or becomes true in the program. It is introduced in order to execute a sequence of virtual updates that is needed due to interacting DECs and ICs. Finally, atoms annotated with $\mathbf{t}^{\star\star}$ are those that become true in a solution. They are the relevant atoms, and are used to read off the database atoms in the solutions (rules 8. below).

The relevant attributes of a constraint are those where the occurrence of *null* is relevant for its satisfaction [11], and then, they receive a special treatment in the logic programs. For the DEC in $\Sigma(\mathtt{P4}, \mathtt{P3})$ in Example 1, the two attributes of R^3 and the first two of R^4 are relevant, but not the third attribute of R^4.

Definition 8. For a constraint $\psi \in \mathcal{L}(\mathcal{R})$ and a variable or a domain constant t, $pos^R(\psi, t)$ is the set of positions in predicate $R \in \mathcal{R}$ where t appears in ψ. The set of *relevant variables* for ψ is $\mathcal{V}(\psi) = \{x \mid x \text{ is a repeated variable in } \psi\}$. The set of *relevant attributes* for ψ is $\mathcal{A}(\psi) = \{R[i] \mid x \in \mathcal{V}(\psi) \text{ and } i \in pos^R(\psi, x)\} \cup \{R[i] \mid c \text{ is a constant in } \psi \text{ and } i \in pos^R(\psi, c)\}$, where $R[i]$ denotes the attribute in position i in R. □

Definition 9. Consider a PDES $\mathfrak{P}$, a peer $\mathtt{P} \in \mathcal{P}$ with $\mathcal{N}(\mathtt{P}) = \{\mathtt{P}, \mathtt{P1}, \ldots, \mathtt{Pn}\}$, and $\mathcal{I} = \{I_1, \ldots, I_n\}$, where I_j is a database instance over the schema of Pj. The *solution program* $\Pi(\mathfrak{P}, \mathtt{P}, \mathcal{I})$ for P contains:

1. $dom(a)$, for every $a \in (\mathcal{U} \setminus \{null\})$. $R(\bar{a})$, for each atom $R(\bar{a}) \in D(\mathtt{P})$. $R(\bar{a})$, for each $R(\bar{a}) \in I$ with $I \in \mathcal{I}$.
2. For every UDEC $\psi \in \Sigma(\mathtt{P}, \mathtt{Pj})$ of the form (1) with $\mathtt{Pj} \in \mathcal{N}(\mathtt{P})$ and $(\mathtt{P}, \{\mathit{same} \text{ or } \mathit{less}\}, \mathtt{Pj}) \in \mathit{trust}$, the rule:

$$\bigvee_{R \in R_{\mathtt{P}}} R(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{Q \in Q_{\mathtt{P}}} Q\_(\bar{y}_j, \mathbf{t_a}) \leftarrow \bigwedge_{i=1}^{n} R_i\_(\bar{x}_i, \mathbf{t}^\star), \bigwedge_{j=1}^{m} Q_j\_(\bar{y}_j, \mathbf{f}^\star), \bigwedge_{x_l \in \mathcal{A}(\psi)} x_l \neq null, \ \bar{\varphi},$$

where $\mathcal{A}(\psi)$ is the set of relevant attributes of ψ, $\bar{\varphi}$ is a conjunction of built-ins that is equivalent to the negation of φ; and, given $\mathcal{R} = \{R_i \mid i = 1, \ldots, n,\ R_i$ appears in (1)$\}$, $R_{\mathtt{P}}$ is defined by $R_{\mathtt{P}} = \mathcal{R} \cap \mathcal{R}(\mathtt{P})$ if $(\mathtt{P}, \mathit{less}, \mathtt{Pj}) \in \mathit{trust}$; and $R_{\mathtt{P}} = \mathcal{R}$ if $(\mathtt{P}, \mathit{same}, \mathtt{Pj}) \in \mathit{trust}$. $Q_{\mathtt{P}}$ is defined analogously in terms of the Q_j predicates in (1).

3. For every RDEC $\psi \in \Sigma(\mathtt{P}, \mathtt{Pj})$ of the form (2) such that $\mathtt{Pj} \in \mathcal{N}(\mathtt{P})$ and $(\mathtt{P}, \{\mathit{same}$ or $\mathit{less}\}, \mathtt{Pj}) \in \mathit{trust}$:
   (a) If $(\mathtt{P}, \mathit{same}, \mathtt{Pj}) \in \mathit{trust}$, the rule:
   $R_{\_}(\bar{x}, \mathbf{f_a}) \vee Q_{\_}(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow R_{\_}(\bar{x}, \mathbf{t}^\star),\ \mathit{not}\ \mathit{aux}_\psi(\bar{x}'),\ \bar{x}' \neq \mathit{null}.$
   (b) If $(\mathtt{P}, \mathit{less}, \mathtt{Pj}) \in \mathit{trust}$ and $R \in \mathcal{R}(\mathtt{P})$, the rule:
   $R_{\_}(\bar{x}, \mathbf{f_a}) \leftarrow R_{\_}(\bar{x}, \mathbf{t}^\star),\ \mathit{not}\ \mathit{aux}_\psi(\bar{x}'),\ \bar{x}' \neq \mathit{null}.$
   (c) If $(\mathtt{P}, \mathit{less}, \mathtt{Pj}) \in \mathit{trust}$ and $Q \in \mathcal{R}(\mathtt{P})$, the rule:
   $Q_{\_}(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow R_{\_}(\bar{x}, \mathbf{t}^\star),\ \mathit{not}\ \mathit{aux}_\psi(\bar{x}'),\ \bar{x}' \neq \mathit{null}.$
   Plus the auxiliary rules:
   $\mathit{aux}_\psi(\bar{x}') \leftarrow Q(\bar{x}', \overline{null}),\ \mathit{not}\ Q_{\_}(\bar{x}', \overline{null}, \mathbf{f_a}),\ \bar{x}' \neq \mathit{null}.$
   For every $y_i \in \bar{y}$:
   $\mathit{aux}_\psi(\bar{x}') \leftarrow Q_{\_}(\bar{x}', \bar{y}, \mathbf{t}^\star),\ \mathit{not}\ Q_{\_}(\bar{x}', \bar{y}, \mathbf{f_a}),\ \bar{x}' \neq \mathit{null}, y_i \neq \mathit{null}.$
4. For every UIC $\psi \in \mathit{IC}(\mathtt{P})$ of the form (1), the rule:
   $$\bigvee_{i=1}^{n} P_{i\_}(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} Q_{j\_}(\bar{y}_j, \mathbf{t_a}) \ \leftarrow\ \bigwedge_{i=1}^{n} P_{i\_}(\bar{x}_i, \mathbf{t}^\star), \bigwedge_{j=1}^{m} Q_{j\_}(\bar{y}_j, \mathbf{f}^\star),\ \bigwedge_{x_l \in \mathcal{A}(\psi)} x_l \neq \mathit{null},\ \bar{\varphi}.$$
5. For every RIC $\psi \in \mathit{IC}(\mathtt{P})$ of the form (2), the rules:
   $P_{\_}(\bar{x}, \mathbf{f_a}) \vee Q_{\_}(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow P_{\_}(\bar{x}, \mathbf{t}^\star),\ \mathit{not}\ \ \mathit{aux}_\psi(\bar{x}'),\ \bar{x}' \neq \mathit{null}.$
   $\mathit{aux}_\psi(\bar{x}') \leftarrow Q(\bar{x}', \overline{null}),\ \mathit{not}\ Q_{\_}(\bar{x}', \overline{null}, \mathbf{f_a}),\ \bar{x}' \neq \mathit{null}.$
   For every $y_i \in \bar{y}$:
   $\mathit{aux}_\psi(\bar{x}') \leftarrow Q_{\_}(\bar{x}', \bar{y}, \mathbf{t}^\star),\ \mathit{not}\ Q_{\_}(\bar{x}', \bar{y}, \mathbf{f_a}),\ \bar{x}' \neq \mathit{null}, y_i \neq \mathit{null}.$
6. For each predicate $R \in \mathcal{R}(\mathcal{N}(\mathtt{P}))$, the annotation rules:
   $R_{\_}(\bar{x}, \mathbf{f}^\star) \leftarrow \mathit{dom}(\bar{x}),\ \mathit{not}\ R(\bar{x}).$ $\qquad$ $R_{\_}(\bar{x}, \mathbf{f}^\star) \leftarrow R_{\_}(\bar{x}, \mathbf{f_a}).$
   $R_{\_}(\bar{x}, \mathbf{t}^\star) \leftarrow R(\bar{x}).$ $\qquad$ $R_{\_}(\bar{x}, \mathbf{t}^\star) \leftarrow R_{\_}(\bar{x}, \mathbf{t_a}).$
7. For each predicate $R \in \mathcal{R}(\mathcal{N}(\mathtt{P}))$, the program constraint:
   $\leftarrow\ R_{\_}(\bar{x}, \mathbf{t_a}), R_{\_}(\bar{x}, \mathbf{f_a}).$
8. For each predicate $R \in \mathcal{R}(\mathtt{P})$, the interpretation rule:
   $R_{\_}(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow R_{\_}(\bar{x}, \mathbf{t}^\star),\ \mathit{not}\ R_{\_}(\bar{x}, \mathbf{f_a}).$ □

In bodies of rules associated to DECs or ICs ψ, the conditions of the form $x \neq \mathit{null}$, with x a variable appearing in a relevant attribute of ψ are used to capture the semantics of null values as used in SQL (cf. [11] for details). An atom of the form $P(\bar{x}, \overline{null}, ...)$ in the program represents an atom with possibly several occurrences of *null*, not necessarily in its last arguments, e.g. $P(x, null, y, null, ...)$. For $\bar{x} = x_1, \ldots, x_n,\ \bar{x} \neq null$ abbreviates $x_1 \neq null, \ldots, x_n \neq null$. The program constraints in 7. discard models where an atom is both inserted and deleted.

The facts of the program are those in instance $D(\mathtt{P})$ of $\mathtt{P}$ and those in instances $I(\mathtt{Pi})$ for $\mathtt{P}$'s neighbors $\mathtt{Pi}$. The instances $I(\mathtt{Pi})$ used in the program may not coincide with the physical instances $D(\mathtt{Pi})$. Actually, as shown below, if each $I(\mathtt{Pi})$ is the intersection of the solutions of $\mathtt{Pi}$, then the stable models of the program are in one-to-one correspondence with the solutions of peer $\mathtt{P}$.

Since virtual updates are executed on the peers' instances, their local ICs have to be kept satisfied. That is the role of rules 4. (for universal ICs) and 5.

(for referential ICs) above. We adopt the stable model semantics for the solution programs [23], i.e. their intended models are their stable models.

Example 6. (Example 1 continued) Consider $D(\mathtt{P1}) = \{R^1(a, 2)\}$, and the instance $I_2 = \{R^2(d, 5)\}$ for P2, the neighbor of P1. The solution program $\Pi(\mathfrak{P}, \mathtt{P1}, \{I_2\})$ contains: $dom(a)$. $dom(c)$. ... $R^1(a, 2)$. $R^2(d, 5)$.
$R^1_\_(x, y, \mathbf{t_a}) \leftarrow R^2_\_(x, y, \mathbf{t}^\star)$, $R^1_\_(x, y, \mathbf{f}^\star), x \neq null, y \neq null$.
$R^1_\_(x, y, \mathbf{t}^\star) \leftarrow R^1_\_(x, y, \mathbf{t_a})$. $\quad R^1_\_(x, y, \mathbf{t}^\star) \leftarrow R^1(x, y)$.
$R^1_\_(x, y, \mathbf{f}^\star) \leftarrow R^1_\_(x, y, \mathbf{f_a})$. $\quad R^1_\_(x, y, \mathbf{f}^\star) \leftarrow dom(x), dom(y),\ not\ R^1(x, y)$.
$\leftarrow R^1_\_(x, y, \mathbf{t_a}), R^1_\_(x, y, \mathbf{f_a})$. $\quad R^1_\_(x, y, \mathbf{t}^{\star\star}) \leftarrow R^1_\_(x, y, \mathbf{t}^\star),\ not\ R^1_\_(x, y, \mathbf{f_a})$.
With similar rules to the 2nd-6th for R^2. The first rule makes sure that an R^2-tuple that is not in R^1, is also virtually inserted into R^1. In this case, since P1 trusts P2 more than itself, virtual changes affect only peer P1. □

Example 7. Consider a PDES $\mathfrak{P}$ with $\mathcal{R}(\mathtt{P1}) = \{R^1(\cdot, \cdot)\}$, $D(\mathtt{P1}) = \{R^1(s, t), R^1(a,\ null)\}$, $\mathcal{R}(\mathtt{P2}) = \{R^2(\cdot,\ \cdot)\}$, $D(\mathtt{P2}) = \{R^2(c, d),\ R^2(a, e)\}$, $\Sigma(\mathtt{P1},\ \mathtt{P2}) = \{\forall xy\ (R^2(x, y) \rightarrow \exists z\ R^1(x, z))\}$, $IC(\mathtt{P1}) = \{\forall xyz\ (R^1(x, y) \wedge R^1(x, z) \rightarrow y = z)\}$, $\Sigma(\mathtt{P2}, \mathtt{P1}) = IC(\mathtt{P2}) = \emptyset$, and $trust = \{(\mathtt{P1},\ less,\ \mathtt{P2})\}$. The program $\Pi(\mathfrak{P}, \mathtt{P1}, \{I_2\})$ for P1 needs an instance for P2 that may be different from $D(\mathtt{P2})$, but in this case we choose $I_2 = D(\mathtt{P2})$, obtaining the following program (omitting rules 6., 7.): $dom(a)$. $dom(b)$. ... $R^1(a, null)$. $R^1(s, t)$. $R^2(c, d)$. $R^2(a, e)$.

$$R^1_\_(x, null, \mathbf{t_a}) \leftarrow R^2_\_(x, \mathbf{t}^\star),\ not\ aux(x), x \neq null.$$
$$aux(x) \leftarrow R^1(x, null),\ not\ R^1_\_(x, null, \mathbf{f_a}).$$
$$aux(x) \leftarrow R^1(x, y, \mathbf{t}^\star),\ not\ R^1_\_(x, y, \mathbf{f_a}), x \neq null, y \neq null.$$
$$R^1(x, y, \mathbf{f_a}) \vee R^1(x, z, \mathbf{f_a}) \leftarrow R^1(x, y, \mathbf{t}^\star), R^1(x, z, \mathbf{t}^\star),\ x \neq null,\ y \neq z.$$
$$R^1_\_(x, y, \mathbf{t}^{\star\star}) \leftarrow R^1_\_(x, y, \mathbf{t}^\star),\ not\ R^1_\_(x, y, \mathbf{f_a}).$$

The first rule has the role of satisfying the RDEC by introducing a *null* into R^1. The fourth rule takes care of the local functional dependency. □

The atoms annotated with $\mathbf{t}^{\star\star}$ in a stable model of P's program have predicates of P only. They define a database instance for P. In Example 7, only $R^1_\_$ -atoms become annotated with $\mathbf{t}^{\star\star}$. The program has only one stable model, with associated instance $\{R^1(a, null), R^1(s, t), R^1(c, null)\}$.

Definition 10. The database instance for peer P associated to a stable model $\mathcal{M}$ of program $\Pi(\mathfrak{P}, \mathtt{P}, \mathcal{I})$ is $D_\mathcal{M} = \{R(\bar{a}) \mid R\_(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}\}$. □

Theorem 2. Given a PDES $\mathfrak{P}$, $\mathtt{P} \in \mathcal{P}$, $\mathcal{N}(\mathtt{P}) = \{\mathtt{P}, \mathtt{P1}, \ldots, \mathtt{Pn}\}$, $n \geq 0$, $D(\mathtt{P})$ an instance for P, and $\mathcal{I}^\star = \{I_1, \ldots, I_n\}$ instances for P1, ..., Pn, resp. If $\Sigma \cup IC$ is ref-acyclic and each of the I_i is the intersection of the solution instances for peer Pi, then the instances of the form $D_\mathcal{M}$, where $\mathcal{M}$ is a stable model of $\Pi(\mathfrak{P}, \mathtt{P}, \mathcal{I}^\star)$, are the all and only solution instances for P. □

In Example 7, given that $D(\mathtt{P2})$ is already the only solution instance for P2 (P2 has neither DECs nor local ICs) and $\Sigma(\mathtt{P1}, \mathtt{P2})$ is ref-acyclic, the only solution instance for P is $D = \{R^1(s, t), R^1(a, null), R^1(c, null)\}$. However, if there are ref-cycles, the stable models may correspond to a strict superset of the solutions for a peer (c.f. Example 12 in Appendix A). In this case, post-processing that deletes models corresponding to non-minimal "solutions" is necessary.

Under the assumption that we have already computed the (intersection of the) solution instances for the neighbors of P, the program for P allows us to compute its solution instances. This generates a recursive process that can be applied because $\mathcal{G}(\mathtt{P})$ is acyclic. The terminal peers Pt, i.e. those with no outgoing edges in $\mathcal{G}(\mathtt{P})$, will become the base cases for the recursion. If a peer P' has Pt as one of its neighbors, the instance I_t to be used for Pt in the program for P' is simply $D(\mathtt{Pt})$, Pt's original local instance.

Theorem 2 still holds if peer P, instead of collecting the intersection I_i of the solutions of a neighbor Pi, uses the intersection of the solutions for Pi restricted to the subschema of Pi that contains Pi's relations that appear in $\Sigma(\mathtt{P}, \mathtt{Pi})$, which are those P needs to run its program.

With a solution program for P, PCAs for a query $\mathcal{Q}$ posed to P can be obtained by running under the stable model semantics a query program $\Pi(\mathcal{Q})$ that represents the query in combination with $\Pi(\mathfrak{P}, \mathtt{P}, \mathcal{I}^\star)$.

Example 8. (Example 6 continued) In order to obtain the PCAs to the query $\mathcal{Q}_1 : R^1(x, y)$, asking for the tuples in R^1, the rule $ans_1(x, y) \leftarrow R^1(x, y, \mathbf{t}^{\star\star})$ has to be added to $\Pi(\mathfrak{P}, \mathtt{P1}, \{I_2\})$ (assuming that I_2 is the intersection of the solution instances for P2). The ground ans_1-atoms in the intersection of all stable models correspond to the PCAs. For the query $\mathcal{Q}_2 : \exists y R^1(x, y)$, the projection of R^1 on its first attribute, the query rule is $ans_2(x) \leftarrow R^1(x, y, \mathbf{t}^{\star\star})$. □

Our semantics could be naively implemented as follows. When P is posed a query, P has to run its program, for which it needs as facts those in the intersections of the solutions of its neighbors. So, 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 $\Sigma(\mathtt{P}, \mathtt{P'})$. P expects to receive from P' the set of PCAs to $\mathcal{Q}$, because they corresponds to the extension of R in the intersection of solutions for P'. In order to return to P the PCAs to its queries, the neighboring peers have to run their own programs (except for the facts, each peer has a fixed solution program that can be used with any query). As before, they will need PCAs from their own neighbors; etc. This recursion will eventually reach peers that have no DECs (and its local ICs will be satisfied), 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 9. (Example 6 continued) Consider local instances $D(\mathtt{P1}) = \{R^1(a, 2)\}$, $D(\mathtt{P2}) = \{R^2(c, 4), R^2(d, 5)\}$, and $D(\mathtt{P3}) = \{R^3(c, 4)\}$. A user poses the query $\mathcal{Q}_0 : R^1(x, y)$ to P1, expecting its PCAs. To run its program, P1 needs the intersection of the solutions of peer P2. So, P1 sends to P2 the queries $\mathcal{Q}_1^1 : R^2(x, y)$ and $\mathcal{Q}_2^1 : S^2(x, y)$ (actually, P1 does not need the latter, S^2 is not relevant to P1). In order to peer-consistently-answer these queries, P2 needs from P3 the PCAs to $\mathcal{Q}_3^2 : R^3(x, y)$. Since P3 has no neighboring peers, it returns to P2 the entire extension in its local database of relation R^3: $I_3 = D(\mathtt{P3}) = \{R^3(c, 4)\}$ is given to P2. Now, P2 can run its solution program $\Pi(\mathfrak{P}, \mathtt{P2}, \{I_3\})$, containing: $dom(c)$. $dom(d)$. ... $R^2(c, 4)$. $R^2(d, 5)$. $R^3(c, 4)$.

$$R^2_{\_}(x, y, \mathbf{f_a}) \vee R^3_{\_}(x, y, \mathbf{f_a}) \leftarrow R^2_{\_}(x, y, \mathbf{t}^\star), R^3_{\_}(x, y, \mathbf{t}^\star), x \neq null, y \neq null.$$
$$R^2_{\_}(x, y, \mathbf{t}^{\star\star}) \leftarrow R^2_{\_}(x, y, \mathbf{t}^\star),\ not\ R^2_{\_}(x, y, \mathbf{f_a}).$$
$$S^2_{\_}(x, y, \mathbf{t}^{\star\star}) \leftarrow S^2_{\_}(x, y, \mathbf{t}^\star),\ not\ S^2_{\_}(x, y, \mathbf{f_a}).$$

We obtain two solutions: $\{\{R^2(d,5)\}$ and $\{R^2(c,4), R^2(d,5)\}\}$. So, the intersection of P2's solutions is $I_2 = \{R^2(d,5)\}$. Finally, the program $\Pi(\mathfrak{P}, \mathtt{P1}, \{I_2\})$ given in Example 6 is run. It has only one solution, namely $\{R^1(a,2), R^1(d,5)\}\}$. Therefore, the peer consistent answers to $\mathcal{Q}_0$ are $(a,2)$ and $(d,5)$. □

4 Discussion

The domain predicate, *dom*, in the solution programs can always be instantiated in a finite active domain; actually *dom* can be eliminated by adding rules [10].

In the most common PDESs, let us call them the *unrestricted import case*, peers P only import data from other peers they trust more than themselves; and using DECs $\Sigma(\mathtt{P})$ of the form (1) or (2) that have only one database predicate in the consequent which belongs to $\mathcal{R}(\mathtt{P})$, and all predicates in the antecedent belonging to another peer's schema. In this case, the relevant part of the intersection of the solutions of each neighbor can be obtained as the PCAs to a single conjunctive query, namely the one in the antecedent of the DEC.

Proposition 1. For the unrestricted import case of PDES, a peer P with an empty set $IC(\mathtt{P})$ of local ICs always has a solution instance. □

This result still holds under rather weak conditions, e.g. if the ICs in $IC(\mathtt{P})$: (a) have a consequent that contains at least one database predicate (not a built-in); or (b) if only built-ins appear in the consequent, e.g. **false**, there is a predicate in the antecedent of the IC that does not appear in any of P's DECs.

If a PDES has no local ICs, then it is easy to see that the solution program is head-cycle free [17], and we obtain

Proposition 2. In the unrestricted import case, the solution program for a peer with an empty set of local ICs is equivalent to a non-disjunctive program. □

The hypothesis on local ICs in this result can be much weakened by assuming that local ICs are of the form identified in [5, 11], that lead to head-cycle free repair programs. Since cautious reasoning from normal logic programs is *coNP*-complete [17], peer consistent answering is in *coNP* in data complexity.

We have assumed that $\mathcal{G}(\mathfrak{P})$ is acyclic. However, the peers, not being aware of being in a cyclic $\mathcal{G}(\mathfrak{P})$, could attempt to do data exchange as described above. In order not to detect an infinite loop, for each query a unique identifier can be created and kept in all the queries that have origin in it.

The assumption of acyclicity of the accessibility graph is quite cautious in the sense that it excludes cases where a reasonable semantics could be given and the logic programs would work correctly, because the cycles in $\mathcal{G}(\mathfrak{P})$ are not relevant (cf. Example 11 in Appendix A).

We have also assumed that the sets $IC(\mathtt{P})$ of local ICs of peers P are each ref-acyclic. Even under this assumption, and also with $\Sigma(\mathtt{P})$ ref-acyclic, $IC(\mathtt{P}) \cup \Sigma(\mathtt{P})$ can have ref-cycles. For example, with $IC(\mathtt{P1}) = \{\forall x(R^1(x) \rightarrow S^1(x))\}$, $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall x(S^1(x) \rightarrow \exists y R^2(x,y)), \forall x(R^2(x,y) \rightarrow R^1(x))\}$. There are also cases with an acyclic $\mathcal{G}(\mathfrak{P})$, but with ref-cycles in the DECs, where the logic programming counterpart of the semantics is correct due to the role of the trust relationships (cf. Example 13 in Appendix A).

It becomes clear that it is possible to find more relaxed conditions, both on the accessibility graph and ref-cycles, under which a sensible semantics for

solutions and semantically corresponding logic programs can be given. Also, with general cyclic accessibility graphs, *super peers* [31] could be used to detect cycles and prune certain DECs, making the graph acyclic if necessary; and then our semantics could be applied.

In [13, 15, 14], the semantics of a PDES is given in terms of epistemic logic. The mappings (our DECs) are of the form $cq_i \rightarrow cq_j$, with cq_i and cq_j conjunctive queries over `Pi` and `Pj`'s schemas, resp. Those DECs keep the schemas separate. It is implicitly assumed that peers trust themselves less than other peers. The semantics can be applied in the presence of cycles in the accessibility graph.

The treatment of local ICs differs from ours in two ways : (a) A peer that is inconsistent wrt its local ICs is not considered for data exchange, while in our case such a peer may apply a repair semantics, as in CQA. (b) Atoms are imported into a peer by interaction with other peers only if this does not produce a local IC violation. In our case, under the same trust conditions, the data is accepted and the peer applies again a local repair semantics.

In order to answer a query [14], a peer traverses the network eventually collecting at its site all DECs, ICs and data of other logically related peers. With these elements, the peer can construct its epistemic theory, that is used for query answering. An accessibility cycle can be detected by using request identifiers. The use of epistemic logic makes sure that *certain data*, the one a peer really knows, is passed to another peer. In our case, a peer collects only data from its neighbors; and certainty is achieved by using the PCAs of a peer, or more generally, the intersection of its solutions. A more detailed comparison can be found in [10].

The semantics in [20, 21] coincides with the epistemic semantics in [15]. They provide a distributed algorithm, where peers' data is updated by instruction of a super peer. When a query is posed to a peer, it can answer the query right away with its data because the PDES is already updated.

5 Conclusions

We have introduced a framework for peer data exchange with trust relationships. Each peer solves its data and semantic conflicts at query time, when querying its own and other peers' data.

Logic programs can be used to specify solutions for a peer and to obtain peer consistent answers. Techniques to partially compute the solution instances can be useful, since we are not interested in them per se, but in the PCAs. Techniques used in CQA, such as magic sets for stable model semantics [19] and identification of predicates that are relevant to queries and constraints, could also be used in this setting, to restrict the number of rules and the amount of data that are needed to run the program [16, 18].

The problem of query evaluation from disjunctive programs is Π_2^P-complete [17], which matches the complexity of PCA. In spite of this, it is possible to identify syntactic classes of PDESs for which peer consistent query answering has a lower complexity, and specifically tailored mechanisms to solve this problem could be developed, as for CQA (cf. [9] for a survey).

The concepts and results presented in this paper smoothly extend the semantics for *local solutions* for a peer as introduced in [7] to the transitive case.

Basically, those local solutions correspond to the neighborhood solutions we introduced above. No general solution programs were presented in [7].

Semantics for PDESs have been introduced and analyzed in [26, 20, 28, 27, 21, 13, 15, 14, 22], but without considering trust relationships. In them, if there is a DEC from `P` to `Q`, it is implicitly assumed that `P` trusts itself less than `Q`. Also, all the research so far, has concentrated on the unrestricted import case. In our setting, a DEC may also restrict the data that can belong to a peer.

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 work was done when L. Bertossi was visiting the Database Group at Edinburgh University. Their hospitality is much appreciated.

References

[1] Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM Symposium on Principles of Database Systems (PODS'99)*. ACM Press, 1999, pp. 68-79.

[2] Arenas, M., Bertossi, L., Chomicki, J. Answer Sets for Consistent Query Answers. *Theory and Practice of Logic Programming*, 2003, 3(4&5):393-424.

[3] Baral, C. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[4] Barcelo, P. and Bertossi, L. Logic Programs for Querying Inconsistent Databases. In *Proc. Practical Aspects of Declarative Languages (PADL'03)*. Springer LNCS 2562, 2003, pp. 208-222.

[5] Barceló, P., Bertossi, L. and 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. 7-33.

[6] Bertossi, L. and Bravo, L. Consistent Query Answers in Virtual Data Integration Systems. In *Inconsistency Tolerance*. Springer LNCS 3300, 2004, pp. 42-83.

[7] Bertossi, L. and Bravo, L. Query Answering in Peer-to-Peer Data Exchange Systems. In *Proc. EDBT Workshop on Peer-to-Peer Computing and Databases (P2P&DB'04)*. Springer LNCS 3268, 2004, pp. 476-485.

[8] Bertossi, L. and Chomicki, J. Query Answering in Inconsistent Databases. In *Logics for Emerging Applications of Databases*. Springer, 2003, pp. 43-83.

[9] Bertossi, L. Consistent Query Answering in Databases. *ACM Sigmod Record*, June 2006, 35(2):68-76.

[10] Bravo, L. Handling Inconsistency in Databases and Data Integration Systems. PhD. Thesis, Carleton University, Department of Computer Science, 2007. http://homepages.inf.ed.ac.uk/lbravo/Publications.htm

[11] Bravo, L., Bertossi, L. Semantically Correct Query Answers in the Presence of Null Values. In *Proc. EDBT WS on Inconsistency and Incompleteness in Databases (IIDB'06)*. Springer LNCS 4254, 2006, pp. 336-357.

[12] Calì, A., Lembo, D. and Rosati, R. 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.

[13] Calvanese, D., De Giacomo, G., Lenzerini, M. and Rosati, R. Logical Foundations of Peer-To-Peer Data Integration. In *Proc. ACM Symposium on Principles of Database Systems (PODS'04)*. ACM Press, 2004, pp. 241-251.

[14] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M. and Rosati, R. Inconsistency Tolerance in P2P Data Integration: An Epistemic Logic Approach. In *Proc. International Symposium on Database Programming Languages (DBPL'05)*. Springer LNCS 3774, 2005, pp. 90-105.

[15] Calvanese, D., Damaggio, E., De Giacomo, G., Lenzerini, M. and Rosati, R. Semantic Data Integration in P2P Systems. In *Proc. VLDB International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'03)*. Springer LNCS 2944, 2004, pp. 77-90.

[16] Caniupan, M. and Bertossi, L. Optimizing Repair Programs for Consistent Query Answering. In *Proc. International Conference of the Chilean Computer Science Society (SCCC'05)*. IEEE Computer Society Press, 2005, pp. 3-12.

[17] Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 2001, 33(3): 374-425.

[18] Eiter, T., Fink, M., Greco, G. and Lembo, D. Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In *Proc. 19th International Conference on Logic Programming (ICLP 03)*. Springer LNCS 2916, 2003, pp. 163-177.

[19] Faber W., Greco G., Leone N. Magic Sets and their Application to Data Integration". *J. Comp. and Sys. Sciences*, 2007, 73(4):584-609.

[20] Franconi, E., Kuper, G., Lopatenko, A. and Serafini, L. A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems. In *Proc. VLDB Workshop on Databases, Information Systems and P2P Computing (DBISP2P'03)*. Springer LNCS 2944, 2004, pp. 64-76.

[21] Franconi, E., Kuper, G. Lopatenko, A. and Zaihrayeu, I. A Distributed Algorithm for Robust Data Sharing and Updates in P2P Database Networks. In *Proc. EDBT Workshop on Peer-to-peer Computing and Databases (P2P&DB'04)*, Springer LNCS 3268, 2004, pp. 446-455.

[22] Fuxman, A., Kolaitis, Ph., Miller, R. and Tan, W. Peer Data Exchange. *ACM Trans. Database Systems*, 2006, 31(4): 1454-1498.

[23] Gelfond, M. and Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365-385.

[24] Gelfond, M. and Leone, N. Logic Programming and Knowledge Representation: The A-Prolog Perspective. *Artificial Intelligence*, 2002, 138(1-2):3-38.

[25] Greco, G., Greco, S. and Zumpano, E. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2003, 15(6):1389-1408.

[26] Halevy, A., Ives, Z., Suciu, D. and Tatarinov, I. Schema Mediation in Peer Data Management Systems. In *Proc. International Conference on Data Engineering (ICDE'03)*. IEEE Computer Society, 2003, pp. 505-518.

[27] Halevy, A., Ives, Z., Madhavan, J., Mork, P., Suciu, D. and Tatarinov, I. The Piazza Peer Data Management System. *IEEE Transactions on Knowledge and Data Engineering*, 2004, 16(7):787-798.

[28] Kementsietsidis, A., Arenas, M., and Miller, R. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *Proc. ACM International Conference on Management of Data (SIGMOD '03)*. ACM Press, 2003, pp. 325-336.

[29] Kolaitis, Ph. Schema Mappings, Data Exchange, and Metadata Management. In *Proc. of ACM Symposium on Principles of Database Systems (PODS '05)*. ACM Press, 2005, pp. 61-75.

[30] Levene, M. and Loizou, G. Null Inclusion Dependencies in Relational Databases. *Information and Computation*, 1997, 136(2):67-108.

[31] Yang, B. and Garcia-Molina, H. Designing a Super-Peer Network. In *Proc. International Conference on Data Engineering (ICDE'03)*, IEEE Computer Society, 2003, p. 49.

Appendix

A Examples

*Example 10 (***A peer may have no solutions).** Consider the PDMS $\mathfrak{P}$ with
P1: $\mathcal{R}(\mathtt{P1}) = \{R^1(\cdot,\cdot)\}$, $D(\mathtt{P1}) = \{\}$,
P2: $\mathcal{R}(\mathtt{P2}) = \{R^2(\cdot,\cdot)\}$, $D(\mathtt{P2}) = \{R^2(a,b)\}$,
P3: $\mathcal{R}(\mathtt{P3}) = \{R^3(\cdot,\cdot), S^3(\cdot)\}$, $D(\mathtt{P3}) = \{R^3(a,b)\}$,
$\Sigma(\mathtt{P1},\mathtt{P2}) = \{\forall x\forall y(R^2(x,y) \rightarrow R^1(x,y))\}$,
$\Sigma(\mathtt{P1},\mathtt{P3}) = \{\forall x\forall y\forall z(R^1(x,y) \wedge R^3(x,y) \rightarrow S^3(x))\}$.
The trust relationships are shown in the accessibility graph in Figure 2. Since P1 trusts the other peers more than itself, it will modify only its own data to enforce the satisfaction of the DECs. The DEC in $\Sigma(\mathtt{P1},\mathtt{P2})$ enforces that (a,b) has to be added to relation R^1. However, $\Sigma(\mathtt{P1},\mathtt{P3})$ forbids its addition. Therefore, peer P1 has no solutions. □

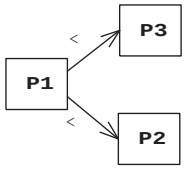


Fig. 2. Accessibility graph of Example 10

*Example 11 (***Cyclic accessibility graph, but reasonable semantics).** Consider $\mathcal{R}(\mathtt{P1}) = \{R^1(\cdot), S^1(\cdot)\}$, $\mathcal{R}(\mathtt{P2}) = \{R^2(\cdot), S^2(\cdot)\}$, $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall x(R^2(x) \rightarrow R^1(x))\}$, $\Sigma(\mathtt{P2},\mathtt{P1}) = \{\forall x(S^1(x) \rightarrow S^2(x))\}$, *trust* = {(P1, *less*, P2), (P2, *less*, P1)}. If a query is posed to P1, it will request from P2 the PCAs to query $R^2(x)$, but not those to query $S^2(x)$. P2 will realize it does not need data from P1 and will return $D(\mathtt{P2})|\{R^2\}$ to P1, who will run its solution program and answer the original query. Even though there is a cycle in $\mathcal{G}(\mathfrak{P})$, there is no infinite loop. □

*Example 12 (***Ref-cycles, complete but unsound program).** For $D(\mathtt{P1}) = \{R^1(a,b)\}$, $D(\mathtt{P2}) = \{R^2(a,c)\}$, $\Sigma(\mathtt{P1}) = \{\forall xz(R^1(x,z) \rightarrow \exists y\ R^2(x,y)), \forall xz(R^2(x,z) \rightarrow \exists y\ R^1(x,y))\}$, $\Sigma(\mathtt{P2}) = \emptyset$; and $(\mathtt{P1}, \mathit{same}, \mathtt{P2}) \in \mathit{trust}$. The set of DECs is ref-cyclic, and even though the only solution for P1 is $\{R^1(a,b)\}$, the associated solution program gives two models. The following are the relevant rules and facts of the solution program:

$dom(a).\quad dom(b).\quad \ldots \quad R^1(a,b).\quad R^2(a,c).$

$$R^1_{\text{-}}(x,y,\mathbf{f_a}) \vee R^2_{\text{-}}(x,null,\mathbf{t_a}) \leftarrow R^1_{\text{-}}(x,y,\mathbf{t}^\star),\ not\ aux_1(x), x \neq null.$$
$$aux_1(x) \leftarrow R^2(x,null),\ not\ R^2_{\text{-}}(x,null,\mathbf{f_a}).$$
$$aux_1(x) \leftarrow R^2(x,y,\mathbf{t}^\star),\ not\ R^2_{\text{-}}(x,y,\mathbf{f_a}), x \neq null, y \neq null.$$
$$R^2_{\text{-}}(x,y,\mathbf{f_a}) \vee R^1_{\text{-}}(x,null,\mathbf{t_a}) \leftarrow R^2_{\text{-}}(x,y,\mathbf{t}^\star),\ not\ aux_2(x), x \neq null.$$
$$aux_2(x) \leftarrow R^1(x,null),\ not\ R^1_{\text{-}}(x,null,\mathbf{f_a}).$$
$$aux_2(x) \leftarrow R^1(x,y,\mathbf{t}^\star),\ not\ R^1_{\text{-}}(x,y,\mathbf{f_a}), x \neq null, y \neq null.$$

The instances obtained from the stable models of this program are $\{R^1(a,b)\}$ and $\emptyset$ which correspond to the neighborhood instances $\{R^1(a,b), R^(a,c)\}$ and

$\emptyset$ respectively. Only the first instance corresponds to a solution instance. The second model is the result of cycles through weak negation (*not*). The cycle creates the self justification of facts as follows: (i) If we choose $R^2_-(a, c, \mathbf{f_a})$ to be true, then by the second and third rule, $aux_1(a)$ is false. (ii) Then, the first rule can be satisfied, by making $R^1_-(a, b, \mathbf{f_a})$ true. (iii) By rules five and six, $aux_2(a)$ is false. (iv) This justifies making $R^2_-(a, b, \mathbf{f_a})$ true, thus, closing the cycle. Notice, that in the whole justification the changes where not determined by inconsistencies. □

*Example 13 (***Ref-cycles, complete and sound program).** (example 12 continued) If we replace $(\texttt{P1}, \mathit{same}, \texttt{P2}) \in \mathit{trust}$ by $(\texttt{P1}, \mathit{less}, \texttt{P2}) \in \mathit{trust}$, rules 1. to 3. of the solution program for peer P1 are: $dom(a)$. $dom(b)$. ... $R^1(a, b)$. $R^2(a, c)$.

$$
\begin{aligned}
R^1_-(x, y, \mathbf{f_a}) &\leftarrow R^1_-(x, y, \mathbf{t}^\star),\ \mathit{not}\ aux_1(x), x \neq \mathit{null}.\\
aux_1(x) &\leftarrow R^2(x, \mathit{null}),\ \mathit{not}\ R^2_-(x, \mathit{null}, \mathbf{f_a}).\\
aux_1(x) &\leftarrow R^2(x, y, \mathbf{t}^\star),\ \mathit{not}\ R^2_-(x, y, \mathbf{f_a}), x \neq \mathit{null}, y \neq \mathit{null}.\\
R^1_-(x, \mathit{null}, \mathbf{t_a}) &\leftarrow R^2_-(x, y, \mathbf{t}^\star),\ \mathit{not}\ aux_2(x), x \neq \mathit{null}.\\
aux_2(x) &\leftarrow R^1(x, \mathit{null}),\ \mathit{not}\ R^1_-(x, \mathit{null}, \mathbf{f_a}).\\
aux_2(x) &\leftarrow R^1(x, y, \mathbf{t}^\star),\ \mathit{not}\ R^1_-(x, y, \mathbf{f_a}), x \neq \mathit{null}, y \neq \mathit{null}.
\end{aligned}
$$

Since P1 trusts more peer P2 than itself, it will modify only its own data. This program computes exactly the solutions for peer P1, i.e. $\{R^1(a, b)\}$, even though the DECs exhibit ref-cycles. □

B Satisfaction of ICs in Databases with NULL

In [11] and [10, chapter 4], a precise, uniform, and logic-based definition of IC satisfaction in SQL databases is given. Loosely speaking, in a commercial DBMS like IBM DB2, a constraint is satisfied if any of the relevant attributes in it has *null* or the constraint is satisfied in the traditional way (i.e. as first-order satisfaction with *null* treated as any other constant).

Example 14. Consider a foreign-key constraint $P[A, B] \subseteq R[A, B]$ and D:

P	A	B	C
	a	5	d
	b	*null*	a

R	$\underline{A}$	$\underline{B}$
	a	5
	a	2

DBMSs implement the so-called simple semantics of the SQL standard for satisfaction of ICs, according to which the database D above satisfies the foreign-key constraint. This is because, for every tuple $\bar{t}$ in P, if $\bar{t}[A]$ and $\bar{t}[B]$ are different from *null*, there is a tuple $\bar{t}'$ in R such that $\bar{t}i[A, B] = \bar{t}'[A, B]$. In this case, the attributes that are relevant for checking the satisfaction of the IC are A and B from both P and R.

Now, the version in first-order logic of the IC is a universal integrity constraint: $\psi: \ \forall xyz\ (P(x, y, z) \rightarrow R(x, y))$. Since x and y appear twice in ψ, by Definition 8, $\mathcal{A}(\psi) = \{P[1], R[1], P[2], R[2]\}$. We can see that the value for C is not relevant to check the satisfaction of the constraint, which makes sense, because we only want to make sure that the values in the first two attributes in P also appear in R.

Now, if we try to insert tuple $(c, d, null)$ into P, the DBMS will reject the tuple since none of the attributes that are relevant for checking the constraint are *null*, and there is no tuple (c, d) in R. □

For the formal definition of IC satisfaction, we need some concepts. Given a set of attributes $\mathcal{A}$ and a predicate $P \in \mathcal{R}$, we denote by $P^{\mathcal{A}}$ the predicate P restricted to (or projected onto) the attributes in $\mathcal{A}$. $D^{\mathcal{A}}$ denotes the database D with all its database atoms projected onto the attributes in $\mathcal{A}$, i.e., $D^{\mathcal{A}} = \{P^{\mathcal{A}}(\Pi_{\mathcal{A}}(\bar{t})) \mid P(\bar{t}) \in D\}$, where $\Pi_{\mathcal{A}}(\bar{t})$ is the projection on $\mathcal{A}$ of tuple $\bar{t}$.

Definition 11. A constraint ψ of the form:

$$\forall \bar{x}(\bigwedge_{i=1}^{n} P_i(\bar{x}_i) \longrightarrow \exists \bar{z}(\bigvee_{j=1}^{m} Q_j(\bar{y}_j, \bar{z}_j) \vee \varphi)), \tag{3}$$

is satisfied in the database instance D, denoted $D \models_N \psi$, iff $D^{\mathcal{A}(\psi)} \models \psi^N$, where ψ^N is

$$\forall \bar{x}(\bigwedge_{i=1}^{m} P_i^{\mathcal{A}(\psi)}(\bar{x}_i) \rightarrow (\bigvee_{v_j \in \mathcal{V}(\psi)} v_j = null \vee \bigvee_{j=1}^{n} Q_j^{\mathcal{A}(\psi)}(\bar{y}_j) \vee \varphi)), \tag{4}$$

and $\bar{x} = \cup_{i=1}^{m} \bar{x}_i$. Here, $D^{\mathcal{A}(\psi)} \models \psi^N$ refers to classical first-order satisfaction, where *null* is treated as any other constant in the domain. □

When we use $D \models \psi$ in the main body of this paper, we are really mean $D \models_N \psi$, as just defined. Notice that UICs, RICs, UDECs and RDECs are special cases of Formula (3). We can see from Definition 11 that there are basically two cases for constraint satisfaction: (a) If *null* is in any of the relevant attributes in the antecedent, then the constraint is satisfied. (b) If *null* does not appear in the relevant attributes, then the second disjunct in the consequent of formula (4) has to be checked, i.e., the consequent of the IC restricted to the relevant attributes. This can be done as usual, treating *null* as any other constant.

Example 15. (example 14 continued) In order to check if $D \models_N \forall xyz\ (P(x, y, z) \rightarrow R(x, y))$, we need to determine if $D^{\mathcal{A}(\psi)} \models \forall xy(P^{\mathcal{A}(\psi)}(x, y) \rightarrow (x = null \vee y = null \vee R^{\mathcal{A}(\psi)}(x, y)))$, with:

$D^{\mathcal{A}(\psi)}$:

$P^{\mathcal{A}_1}$	A	B
	a	5
	b	*null*

$R^{\mathcal{A}_1}$	D	E
	a	d

For $x = a$ and $y = 5$, $D^{\mathcal{A}_1} \models P^{\mathcal{A}(\psi)}(a, 5)$, but none of them is *null*, therefore we need to check if $D^{\mathcal{A}(\psi)} \models R^{\mathcal{A}(\psi)}(a, 5)$. This is true, therefore the constraint is satisfied for $x = a$ and $y = 5$. Now, for $x = b$ and $y = null$, $D^{\mathcal{A}(\psi)} \models P^{\mathcal{A}(\psi)}(b, null)$, and since $y = null$, the constraint is satisfied.

If we add tuple $P(c, d, null)$ to D, it would become inconsistent with respect to constraint (a), because $D^{\mathcal{A}(\psi)} \not\models (P^{\mathcal{A}(\psi)}(c, d) \rightarrow (c = null \vee d = null \vee R^{\mathcal{A}(\psi)}(c, d)))$. □

The semantics of IC satisfaction in Definition 11 coincides, extends, and puts on a solid logical ground the notion of IC satisfaction as implemented in DBMSs that follow the SQL standard. More details can be found in [10, chapter 4].