Consistency and Trust in Peer Data Exchange Systems

LEOPOLDO BERTOSSI*
Carleton University, School of Computer Science, Ottawa, Canada.
bertossi@scs.carleton.ca

LORETO BRAVO
Universidad del Desarrollo, Facultad de Ingeniería, Santiago, Chile.
bravo@udd.cl

submitted 4 May 2014; revised 4 September 2015; accepted 4 May 2016

Abstract

We propose and investigate a semantics for *peer data exchange systems* where different peers are related by data exchange constraints and trust relationships. These two elements plus the data at the peers' sites and their local integrity constraints are made compatible via a semantics that characterizes sets of *solution instances* for the peers. They are the intended -possibly virtual- instances for a peer that are obtained through a data repair semantics that we introduce and investigate. The semantically correct answers from a peer to a query, the so-called *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. The repair semantics is based on null values as used in SQL databases, and is also of independent interest for repairs of single databases with respect to integrity constraints.

KEYWORDS: Peer data exchange, answer set programs, disjunctive stable model semantics, metadata management, schema mappings, relational databases, integrity constraints, database repairs, consistency.

1 Introduction

A *peer data exchange system* (PDES) (also known as a peer data management system) can be conceived as a finite set $\mathcal{P} = \{\texttt{P}_1, \ldots, \texttt{P}_n\}$ of peers, each of them owning a local relational schema and a database instance. When a peer $\texttt{P}$ receives a query, in order to answer it, $\texttt{P}$'s data is completed or modified according to relevant data that other peers may have. For this to be possible, $\texttt{P}$ has to be directly related to some neighbor peers $\texttt{Q}$ through sets, $\Sigma(\texttt{P}, \texttt{Q})$, of *logical mappings* or *data exchange constraints* (DECs), from $\texttt{P}$ to $\texttt{Q}$. They are first-order sentences expressed in terms

* Contact author.

of the elements of the database schemas of P and Q. In their turn, P's neighbors may have their own neighbors, who through their data may, transitively and eventually, have an impact on P's data as well.

DECs between two peers are expected to be satisfied by the combination of the two local instances. In this sense, DECs act as integrity constraints (ICs) on the combination of two schemas and their instances. However, they are not forced to be satisfied, and there is no mechanism in place for maintaining them. Actually, *it is the inconsistency of a DEC that will enable the movement of data between peers, at query-answering time*: When a peer P receives a query, it examines its DECs, on that basis poses queries to its neighbors, who return *consistent* data to P. P also queries its own database. If its DECs are not satisfied by the collected data, P, after a -possibly virtual- consistency repair process, composes a *consistent* answer to the original query. A peer's answering of a query received from an external user or another peer triggers an iterative process of interleaved consistency checking of DECs with respect to the data at hand, and the repair of the latter if necessary.

This movement of data among peers is used for query answering, but not for updating the local physical instances with the purpose of having the DECs satisfied. Of course, this could be done by a peer who receives data from other peers, if it decides to do so. However, it is not the goal of data exchange in such a system to have all the instances synchronized and globally satisfying the DECs. Peers are autonomous, their instances are possibly being independently updated, and other peers are not expected to be aware of that. DECs constrain and trigger data exchange through their inconsistency, which is detected and resolved locally, by a single peer, namely the one who owns those DECs. *We do not assume any kind of central or global coordination; nor that any two neighbors interact for DEC evaluation.*

A peer P that is answering a query may, at query-answering time, import data from its neighbors, to complement its data and/or ignore part of its own data. The way a peer uses the imported data of course depends on its DECs, but also on the *trust relationships* to its neighbors: A peer P may trust its data the same as or less than a neighbor's data. As a consequence, *in our PDESs, data exchange is driven by inconsistency and constrained by trust.*

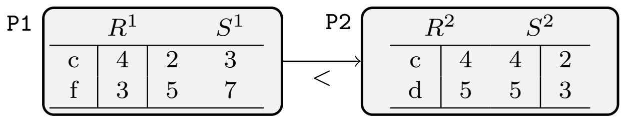
Example 1.1
Peers P1 and P2 have relational schemas $\mathcal{S}(\texttt{P1}) = \{R^1(\cdot,\cdot,\cdot), S^1(\cdot)\}$, and $\mathcal{S}(\texttt{P2}) = \{R^2(\cdot,\cdot), S^2(\cdot,\cdot)\}$. Here, P1 is connected to peer P2 by P1's set of DECs:

$$\Sigma(\texttt{P1},\texttt{P2}) = \{\forall x \forall y (R^2(x,y) \wedge S^2(y,z) \rightarrow R^1(x,y,z)),\ \forall x (S^1(x) \rightarrow S^2(5,x))\}\cdot$$

These DECs belong to P1; and P2 may not even be aware of them. Let's assume that P1 trusts P2 more than itself. The existence of DECs from P1 to P2, and the trust relationship are indicated in Figure 1 by the labeled arrow from one instance to the other.

We can see that the DECs are not satisfied by the combined instance, which is perfectly acceptable; we do not have to do anything.

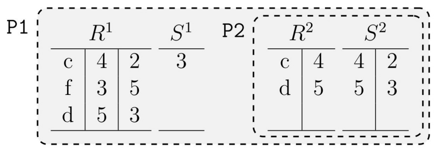
Now, assume that the query $\mathcal{Q}(x) : \exists yz R^1(x,y,z)$ is posed to P1. In order to



R^1			S^1
c	4	2	3
f	3	5	7

R^2		S^2	
c	4	4	2
d	5	5	3

Fig. 1. A simple PDES



R^1			S^1
c	4	2	3
f	3	5	
d	5	3	

R^2		S^2	
c	4	4	2
d	5	5	3

Fig. 2. A neighborhood solution instance for P1

answer it, P1 notices through the DECs that P2's schema is related to its predicate R^1. Accordingly, P1 has to adjust its own data, so that the DECs with P2 are satisfied. In order to do this, P1 requests P2's data, by issuing to P2 atomic queries in P2's language, namely $\mathcal{Q}_1(x, y)\colon R^2(x, y)$ and $\mathcal{Q}_2(x, y)\colon S^2(x, y)$.[1] Notice that, at this stage and from the point of view of P2, the queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ can be considered as queries posed by an external user, so as query $\mathcal{Q}$ was posed to P1.

Since P2 has neither DECs to other peers nor local ICs, it will return to P1 its physical data, without any modification, as query answers. Now P1, confronted with a combined instance, can detect that the first DEC is not satisfied, because the tuple $R^1(d, 5, 3)$ is not in R^1. Nor the second DEC, because the tuple $S^2(5, 7)$ is not in S^2.

Since P1 trusts P2's data more than its own, the inconsistencies are locally solved by P1 by inserting $R^1(d, 5, 3)$ into R^1, and deleting tuple $S^1(7)$ from S^1. These changes could be only virtual, and for the purpose of answering the query at hand. In this case, P1 has a single *neighborhood solution* (instance), that is, an instance for the combined schema that satisfies the DECs, and, in some sense, *minimally departs* from the (inconsistent) instance that had been formed at P1' neighborhood. This neighborhood solution for P1 is shown in Figure 2. The restriction of this instance to P1's schema is *a solution* (instance) for P1, and is used to answer the initial query posed to P1. Thus, the answers to $\mathcal{Q}$ are $\langle c\rangle, \langle f\rangle, \langle d\rangle$. □

In the example, for illustration purposes, we used a particular and well-known *repair semantics*, i.e. a particular way of restoring consistency on a database that does not satisfy a given set of ICs,[2] while still staying "as close as possible" to the given, inconsistent instance. In this case, repairs minimize, under set inclusion, the set of tuples that are deleted or inserted (Arenas et al. 1999). Repair semantics come with their minimality criteria. (Cf. (Bertossi 2011) for a general discussion of repair semantics, and for references to different repair semantics.) In this work we

[1] Actually, P1 does not need the whole extensions of P2's tables. However, asking for P2's whole tables allow us to better illustrate our general approach.
[2] We emphasize that DECs can be seen as ICs on combined schemas and instances.

will introduce, investigate and use a variation of the repair semantics used in the example.

In some cases, a peer P may have several neighborhood solutions, leading to also possibly several (local) solution instances. The answers returned by P (to an external user or another peer) will be the *peer consistent answers*, i.e. those that are shared (or returned) by all the different solution instances for P. That is, a *cautious* (also known as *skeptical* or *certain*) semantics is applied to query answering (Brewka et al. 2011; Leone et al. 2006). The same do P's neighbors, who may have solutions of their own, when they hand over their data to P: They give away their own peer-consistent data.

The notion of solution instance for a peer is used as an auxiliary notion, to characterize the semantically correct answers from P's point of view. When trying to correctly answer a query, we try to avoid, as much as possible, the generation of material solutions instances. Ideally, a peer would compute its peer consistent answers to a user query just by querying the already available local instances, and dealing with the involved DECs on-the-fly, at query answering time.

In this work we formalize these ideas, by first providing a general semantic framework for PDESs and their solution-semantics. It captures in abstract terms, and in this order, the notions of: (a) PDES, including a broad and useful syntactic class of DECs; (b) neighborhood solution for a peer, appealing to an *abstract repair-semantics* that involves trust relationships; (c) solution for a peer, a recursive concept due to a possible (always finite) chain of peers building each, one after another, neighborhood solutions; and (d) peer-consistent answer.

The result is a formal semantics for a system of peers who exchange data for query answering. It is a model-theoretic *possible-world semantics* that characterizes the *class of intended instances for a peer*, the above mentioned *solutions*. The expected answers from a peer are *certain* with respect to that class, i.e. true in all of them.

Next, as a second main subject, we instantiate the general semantic framework, and introduce and investigate a specific repair (or solution) semantics for PDESs. We do so by making some specific commitments and assumptions: (a) Database instances may have null values as those used in SQL relational databases. (b) Those null values behave and are handled as in DBMSs that (at least partially) comply with the SQL Standard, in particular with respect to IC satisfaction and query answering. (c) The null-based solution semantics that we introduce heavily depends on the presence of nulls à la SQL, and on the use of nulls to restore consistency. (d) This null-based repair semantics comes with its own minimality criterion. (e) Query answering, and DEC and IC satisfaction follow a *a logic* in databases with null à la SQL that we formalize for conjunctive queries and a broad class of DECs and ICs.

For the gist, the null-based repair semantics we adopt does not allow, for example, to introduce a null value to satisfy a join, such as that with the existential z in the DEC $\forall x \forall y(R(x,y) \rightarrow \exists z(S(x,z) \wedge T(z,y)))$. In this case, a tuple deletion from R will be preferred. However, a null can be used for variable z in a tuple inserted to satisfy the DEC $\forall x \forall y(R(x,y) \rightarrow \exists z S(x,z))$ (a deletion from R is also possible).

Our decision to consider null values, as used in SQL relational database systems, for single database repairs and solutions in PDESs is based upon and motivated by the facts that: (a) In this way our research becomes close to database practice; (b) it can be made compatible with and applicable in that practice; (c) SQL DBMSs implicitly implement a sort of null-based repair semantics (when maintaining ICs); (d) that "repair semantics" deserves a theoretical investigation; and (e) at least a sizeable part of the SQL standard and practice can be be put on solid logical grounds. Actually, we are able to bring into this research essentially all the classes of integrity constraints and logical mappings that are used in both database practice and research, and to treat them according to the just mentioned null-based SQL semantics.

We emphasize, however, that our abstract framework is sufficiently modular and flexible to adopt alternative repair semantics for dealing with inconsistency and incompleteness of data.

We see our work also as a contribution to the subjects of repairs and consistent query answering for single relational databases. Repairs based on the use of nulls; actually a single one with a semantics in the style of SQL -which we also logically formalize in this work- had been presented in preliminary form in (Bravo and Bertossi 2006; Bravo 2007). Here, we provide an extended and definitive formulation, and we apply it to peer data exchange.

We also investigate complexity-theoretic and algorithmic issues related to the solution semantics. More specifically, we show that deciding if a neighborhood instance is a neighborhood solution for a peer is $coNP$-complete in data, and deciding if a tuple is a peer consistent answer to a local query is Π^P_2-complete in data. We investigate a particular, important and common case of the PDES semantics, where peers just import data from other peers to complete their own data set, without giving semantic priority to the latter. This leads to a reduction in data complexity for the solution checking problem.

Finally, we show that the null-based, model-theoretic semantics for PDESs can be captured by means of disjunctive logic programs with stable model semantics (Gelfond and Lifschitz 1991), also known as *answer set programs* (ASPs) (Brewka et al. 2011). More specifically, we establish an explicit correspondence between the solutions for a peer and the stable models of the program, obtaining, in this way, a declarative semantics for PDESs that can also be made executable.

In relation to most immediately related work, the idea of peer consistent query answering resembles that of *consistent query answering* (CQA) in databases (Arenas et al. 1999): Consistent answers to a query posed to a database that may be inconsistent with respect to certain ICs are invariant under all *repairs*, i.e. the minimally repaired and consistent versions of the original instance. Furthermore, there are mechanisms for computing consistent answers that avoid or minimize the physical generation of repairs. Logic programs for specifying database repairs as their stable models have been proposed and investigated in (Arenas et al. 2003; Greco et al. 2003; Barcelo and Bertossi 2003; Caniupan and Bertossi 2010). Cf. (Bertossi 2011; Bertossi 2006; Chomicki 2007) for recent surveys of CQA.

Our work can be classified among those on *semantic and schema-based approaches to peer-to-peer data management systems* (Halevy et al. 2003; Halevy et al. 2004; Calvanese et al. 2004; Franconi et al. 2004; Bertossi and Bravo 2004b; Calvanese et al. 2005; Fuxman et al. 2006). The emphasis is on specifying the intended and legal database instances behind a system of peers who are connected with each other by means of schema mappings.

Different notions and forms of trust have been considered in P2P information sharing systems (Demolombe 2011; Sabater and Sierra 2005; Artz and Gil 2007; Marti and Garcia-Molina 2006) (cf. the special issue (Boutaba and Marshall 2006)), but not much in the context of semantic PDESs, where the trust relationships between peers are used to essentially modulate or qualify the use of the data exchange constraints between peers. Our emphasis is on the integration of trust and data exchange constraints, with a notion of trust that is closer to the notion of relative reputation or reliability of a peer as a source of information, in relation to the quality of its data (Marti and Garcia-Molina 2006). In this work, trust relationships are given, and not computed or updated (Jøsang et al. 2006; Jøsang et al. 2012).

This paper is structured as follows. Section 2 provides technical preliminaries and basic elements of PDESs. Section 3 introduces a general semantic framework for PDESs. Section 4, in preparation for the introduction of what will be the official, concrete peer-solution semantics, discusses and formalizes database instances with null values as used in SQL. It also formulates query answering and integrity satisfaction in those databases. Section 5 instantiates the abstract semantic framework proposed in Section 3 by introducing a particular consistency restoration semantics based on the use of null values. This section also investigates the complexity of some computational problems. Section 6 captures the semantics of the previous section in terms of logic programs with stable model semantics. In Section 7 we discuss related work. In Section 8 some final conclusions are drawn. The Online Appendix A discusses several additional and alternative approaches and issues with respect to the previously introduced general and specific semantics. The Online Appendix B gives proofs of our main results. This work considerably extends and develops the semantics for PDESs first suggested in (Bertossi and Bravo 2004b), and further developed in (Bertossi and Bravo 2007).

2 The Basic PDES Scenario

Every peer in a PDES will have a local relational schema with a local relational instance. For this reason, we recall first some basic notions from relational databases. A relational schema $\mathcal{S}$ consists of a set of relational predicates. A relational predicate $R \in \mathcal{S}$ with arity n and attributes $A_1, \cdots, A_n$, is commonly denoted with $R(A_1, \ldots, A_n)$, or sometimes simply $R(\cdot, \ldots, \cdot)$. Each attribute A of (a predicate in) $\mathcal{S}$ has a possibly infinite data domain, $\mathit{Dom}(A)$. In general, we will denote with $\mathcal{U}$ the union of the attribute domains, obtaining a single, possibly infinite *data domain*.

In addition to the *database predicates* in a relational schema $\mathcal{S}$, we have a set $\mathcal{B}$ of built-in predicates (that have a fixed semantics), e.g. comparison predicates, such as $=, \neq, <$, etc. We assume that $\mathcal{B}$ contains the propositional predicate **false** that is always false. (Later on, we will introduce some additional, specific, built-in predicates.)

The predicates of the relational schema $\mathcal{S}$ plus those in $\mathcal{B}$ (that we leave implicit), and the constants in $\mathcal{U}$ determine a language $\mathcal{L}(\mathcal{S})$ of first-order predicate logic. The schema may also contain *integrity constraints*, that are sentences of $\mathcal{L}(\mathcal{S})$.

An instance D for a schema $\mathcal{S}$ is a finite set of a ground atoms of the form $R(\bar{c})$, where $R(A_1, \ldots, A_n) \in \mathcal{S}$ has some arity n, and $\bar{c} = \langle c_1, \ldots, c_n \rangle$, with $c_i \in Dom(A_i)$. For each n-ary predicate $R \in \mathcal{S}$, an instance D for $\mathcal{S}$ determines an extension for R that is a finite n-ary relation over the data domain. If $t \in D$ we denote by $t[\bar{A}]$ the sequence of values in $\bar{t}$ for attributes $A \in \bar{A}$. Given an instance D, the *active domain* of D, denoted, $Adom(D)$, is the finite subset of $\bigcup_A Dom(A)$ that contains all the constants that appear in relations in D.

If $\mathcal{S}'$ is a subschema of $\mathcal{S}$, i.e. contains some of the relational predicates in $\mathcal{S}$, and D is an instance for $\mathcal{S}$, then $D \downharpoonright \mathcal{S}'$ denotes the restriction of D to $\mathcal{S}'$, i.e. $D \downharpoonright \mathcal{S}' = \{R(\bar{t}) \mid R \in \mathcal{S}' \text{ and } R(\bar{t}) \in D\}$.

A database instance D for the schema $\mathcal{S}$ serves then as an interpretation for the language $\mathcal{L}(\mathcal{S})$. If D is an instance for $\mathcal{S}$, and Ψ is a set of sentences of $\mathcal{L}(\mathcal{S})$, then $D \models \Psi$ denotes that D satisfies (makes true) all the sentences in Ψ. If Ψ is the set of integrity constraints that comes with the schema $\mathcal{S}$ and $D \models \Psi$, we say that D is *consistent*. Otherwise, D is *inconsistent*.[3]

A query $\mathcal{Q}(\bar{x})$ is a formula of a language $\mathcal{L}(\mathcal{S})$, where $\bar{x}$ is the list of free variables. A sequence of constants $\bar{c}$ is an answer to $\mathcal{Q}(\bar{x})$ in instance D for $\mathcal{S}$ if $D \models \mathcal{Q}[\bar{c}]$, i.e. the formula becomes true in D when the variables in $\bar{x}$ are replaced by the corresponding constants in $\bar{c}$. When $\bar{x}$ is empty, the query $\mathcal{Q}$ is a *Boolean query*, i.e. a sentence. In this case, the answer in D can be *yes* or *no* depending on whether it is true or not in D, denoted $D \models \mathcal{Q}$, resp. $D \not\models \mathcal{Q}$.

Now we introduce some of the elements that will form a *peer data exchange system* (PDES). After introducing them we will give the formal definition of a PDES (cf. Definitions 2.1 and 2.2 below).

A PDES contains a finite set $\mathcal{P} = \{\mathtt{P1}, \ldots, \mathtt{Pn}\}$ of peers. Each peer $\mathtt{P}$ owns a relational database schema $\mathcal{S}(\mathtt{P})$, and a database instance $D(\mathtt{P})$ for the schema $\mathcal{S}(\mathtt{P})$. We denote with $\mathfrak{S}$ the set of all schemas of a PDES, that is, $\mathfrak{S} = \{\mathcal{S}(\mathtt{P1}), \ldots, \mathcal{S}(\mathtt{Pn})\}$. We assume, to simplify the presentation and, without loss of generality, that the peers' schemas are mutually disjoint, but share a common, possibly infinite database domain $\mathcal{U}$. Each $D(\mathtt{P})$ can be seen as a finite set of ground atoms over $\mathcal{U}$, with predicates in $\mathcal{S}(\mathtt{P})$. It holds, $Adom(D(\mathtt{P})) \subseteq \mathcal{U}$.

The peers' schemas, or unions thereof, determine first-order languages, e.g. $\mathcal{L}(\mathtt{P})$, $\mathcal{L}(\mathtt{P}, \mathtt{Q})$, which are $\mathcal{L}(\mathcal{S}(\mathtt{P}))$ and $\mathcal{L}(\mathcal{S}(\mathtt{P}) \cup \mathcal{S}(\mathtt{Q}))$, resp. A *data exchange constraint* (DEC) between peers $\mathtt{P}$, $\mathtt{Q}$ is an $\mathcal{L}(\mathtt{P}, \mathtt{Q})$-sentence. We will consider the following two syntactic classes of DECs:

[3] In this work, whenever we consider sets Σ of integrity constraints, we assume that Σ is *logically consistent*.

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

$$\forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \longrightarrow \bigwedge_{k=1}^{l}(\bigvee_{j=1}^{m} Q_{kj}(\bar{y}_{kj}))), \tag{1}$$

where the R_i are predicates in $\mathcal{S}(\mathtt{P}) \cup \mathcal{S}(\mathtt{Q})$, the Q_{kj} are atomic formulas with predicates in $\mathcal{S}(\mathtt{P}) \cup \mathcal{S}(\mathtt{Q})$ or atoms with predicates in $\mathcal{B}$, $\bar{x} = \bigcup_{i=1}^{n} \bar{x}_i$, and $\bar{y}_{kj} \subseteq \bar{x}$.

(*b*) A *referential data exchange constraint* (RDEC) between peers $\mathtt{P}$, $\mathtt{Q}$ is an $\mathcal{L}(\mathtt{P},\mathtt{Q})$-sentence of the form:

$$\forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \quad\longrightarrow\quad \exists \bar{y}(\bigwedge_{k=1}^{l} Q_k(\bar{x}_k,\bar{y}_k) \wedge \varphi_k(\bar{x}'_k,\bar{y}'_k)) \vee \varphi(\bar{x})), \tag{2}$$

where the R_i, Q_k are predicates in $\mathcal{S}(\mathtt{P}) \cup \mathcal{S}(\mathtt{Q})$, φ_k and φ are a conjunction, resp. a disjunction, of atoms with predicates in $\mathcal{B}$, and $\bar{x}_i, \bar{x}_k \subseteq \bar{x}$, $\bar{x}'_k \subseteq \bar{x}_k$, $\bar{y}'_k \subseteq \bar{y}_k$, and $\bar{y} = \bigcup_{k=1}^{l} \bar{y}_k \neq \emptyset$.

The formulas $\varphi_k(\bar{x}'_k, \bar{y}'_k))$ are used to impose conditions on the existential values, and φ for conditions on the values for $\bar{x}$ in the antecedent.

The classes of exchange constraints that we are considering are broad enough to capture all the relevant integrity constraints and logical mappings found in data exchange and virtual data integration that are usually considered in the theoretical, technical and applied literature on data management. In particular, UDECs can be used to express *equality-generating dependencies* (egds), and RDECs can express general *tuple-generating dependencies* (tgds) (Abiteboul et al. 1995).

Each peer $\mathtt{P}$ of $\mathcal{P}$ has a possibly empty collection of sets of DECs, $\Sigma(\mathtt{P},\mathtt{Q})$, between $\mathtt{P}$ and peers $\mathtt{Q} \in \mathcal{P}$, with at most one $\Sigma(\mathtt{P},\mathtt{Q})$ for each peer $\mathtt{Q}$. Each $\Sigma(\mathtt{P},\mathtt{Q})$ is finite and logically consistent. Due to the local nature of PDESs systems, it is possible for $\Sigma(\mathtt{P},\mathtt{Q})$ (which is owned by $\mathtt{P}$) to be different from $\Sigma(\mathtt{Q},\mathtt{P})$ (which is owned by $\mathtt{Q}$). The DECs in $\Sigma(\mathtt{P},\mathtt{P})$ are the integrity constraints for (instances of) peer $\mathtt{P}$. We denote with Σ the class formed by of all non-empty sets $\Sigma(\mathtt{P},\mathtt{Q})$ of a PDES.

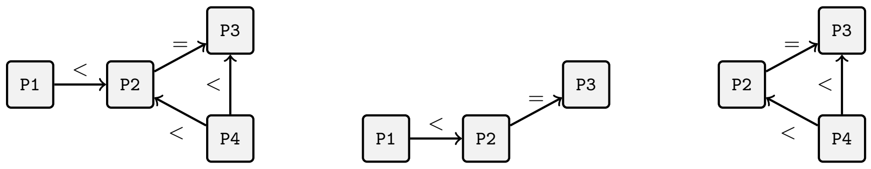
A PDES also has a relation $\mathit{Trust} \subseteq \mathcal{P} \times \{\mathit{less}, \mathit{same}\} \times \mathcal{P}$, with exactly one triple of the form $(\mathtt{P},\cdot,\mathtt{Q})$ for each $\Sigma(\mathtt{P},\mathtt{Q}) \in \Sigma$. The intended semantics of $(\mathtt{P},\mathit{less},\mathtt{Q}) \in \mathit{Trust}$ is that peer $\mathtt{P}$ trusts itself less than $\mathtt{Q}$; while $(\mathtt{P},\mathit{same},\mathtt{Q}) \in \mathit{Trust}$ indicates that $\mathtt{P}$ trusts itself the same as $\mathtt{Q}$.[4] The trust relation is not necessarily symmetric, e.g. it could hold $(\mathtt{P},\mathit{less},\mathtt{Q}), (\mathtt{Q},\mathit{same},\mathtt{P}) \in \mathit{Trust}$. For a peer $\mathtt{P}$, when $\Sigma(\mathtt{P},\mathtt{P}) \neq \emptyset$, we assume $(\mathtt{P},\mathit{same},\mathtt{P}) \in \mathit{Trust}$.

Definition 2.1

(*a*) The *schema of a PDES* $\mathfrak{P}$ is a sequence $\langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$, where $\mathcal{P}$ is a set of peers, $\mathfrak{S}$ is a corresponding set of peer database schemas, Σ the set of DECs, and *Trust* the set of trust relationships.

(*b*) An instance $\mathfrak{D}$ of a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$ is a set containing one database instance $D(\mathtt{P})$ for the schema $S(\mathtt{P}) \in \mathfrak{S}$, for each peer $\mathtt{P} \in \mathcal{P}$. □

[4] We do not consider the case when a peer $\mathtt{P}$ trusts itself more than another peer, since the information of the latter should be irrelevant to $\mathtt{P}$.



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

Fig. 3. Graphs for Example 2.1

Intuitively, and due to the locality of peers, a peer $\mathtt{P}$ will be aware only of the sets $\Sigma(\mathtt{P},\mathtt{Q})$ and elements of *Trust* whose first argument is $\mathtt{P}$. More precisely, a peer stores its own database schema and instance, its DECs to other peers, and its trust relationships. Furthermore, for an instance $\mathfrak{D}$ of schema $\mathfrak{P}$, the instances $D(\mathtt{P}) \in \mathfrak{D}$ are not required to (jointly) satisfy the DECs in Σ. Consistency and consistency restoration is a concern only when queries are posed to peers.

Definition 2.2
Given a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$:

(*a*) The *schema of a peer* $\mathtt{P} \in \mathcal{P}$ is $\mathfrak{P}(\mathtt{P}) = \langle \mathcal{S}(\mathtt{P}), \Sigma(\mathtt{P}), \mathit{Trust}(\mathtt{P})\rangle$, where $\mathcal{S}(\mathtt{P}) \in \mathfrak{S}$, $\Sigma(\mathtt{P}) = \{\varphi \mid \varphi \in \Sigma(\mathtt{P},\mathtt{Q}),\ \Sigma(\mathtt{P},\mathtt{Q}) \in \Sigma, \text{ and } \mathtt{Q} \in \mathcal{P}\}$; and $\mathit{Trust}(\mathtt{P}) = \{(\mathtt{P}, t, \mathtt{Q}) \mid (\mathtt{P}, t, \mathtt{Q}) \in \mathit{Trust}, \text{ and } \mathtt{Q} \in \mathcal{P}\}$. (We can safely identify a peer $\mathtt{P}$ with its schema $\mathfrak{P}(\mathtt{P})$.)

(*b*) The *accessibility graph* $\mathcal{G}(\mathfrak{P})$ contains a vertex for each $\mathtt{P} \in \mathcal{P}$, and a directed edge from $\mathtt{P}$ to $\mathtt{Q}$ if $\mathtt{P} \neq \mathtt{Q}$ and $\Sigma(\mathtt{P},\mathtt{Q}) \neq \emptyset$. An edge from $\mathtt{P}$ to $\mathtt{Q}$ is labeled with "$<$" when $(\mathtt{P}, \mathit{less}, \mathtt{Q}) \in \mathit{Trust}$, and with "$=$" when $(\mathtt{P}, \mathit{same}, \mathtt{Q}) \in \mathit{Trust}$.

(*c*) Peer $\mathtt{P'}$ is *accessible* from $\mathtt{P}$ if there is a path in $\mathcal{G}(\mathfrak{P})$ from $\mathtt{P}$ to $\mathtt{P'}$ or if $\mathtt{P'}{=}\mathtt{P}$. $\mathcal{AC}(\mathtt{P})$ denotes the set of peers accessible from $\mathtt{P}$. $\mathcal{G}(\mathtt{P})$ denotes the restriction of $\mathcal{G}(\mathfrak{P})$ to the peers in $\mathcal{AC}(\mathtt{P})$ (i.e. it contains as vertices only the peers in $\mathcal{AC}(\mathtt{P})$, and the edges between them are those inherited from $\mathcal{G}(\mathfrak{P})$).

(*d*) Peer $\mathtt{P'}$ is a *neighbor* of $\mathtt{P}$ if there is an edge from $\mathtt{P}$ to $\mathtt{P'}$ in $\mathcal{G}(\mathfrak{P})$, or if $\mathtt{P'} = \mathtt{P}$. We denote with $\mathcal{N}(\mathtt{P})$ the set of neighbors of $\mathtt{P}$; and with $\mathcal{S}(\mathcal{N}(\mathtt{P}))$ the union of their schemas, i.e. $\bigcup_{\mathtt{Q}\in\mathcal{N}(\mathtt{P})}\mathcal{S}(\mathtt{Q})$. Furthermore, $\mathcal{N}^\circ(\mathtt{P}) := \mathcal{N}(\mathtt{P}) \smallsetminus \{\mathtt{P}\}$. □

Example 2.1
Consider $\mathfrak{P} = \langle \mathcal{P}, \mathcal{S}, \Sigma, \mathit{Trust}\rangle$ with $\mathcal{P} = \{\mathtt{P1},\mathtt{P2},\mathtt{P3},\mathtt{P4}\}$, $\mathfrak{S} = \{\mathcal{S}(\mathtt{P1}), \mathcal{S}(\mathtt{P2}), \mathcal{S}(\mathtt{P3}), \mathcal{S}(\mathtt{P4})\}$, $\mathcal{S}(\mathtt{P1}) = \{R^1(\cdot,\cdot)\}$, $\mathcal{S}(\mathtt{P2}) = \{R^2(\cdot,\cdot),\ S^2(\cdot,\cdot)\}$, $\mathcal{S}(\mathtt{P3}) = \{R^3(\cdot,\cdot)\}$, $\mathcal{S}(\mathtt{P4}) = \{R^4(\cdot,\cdot,\cdot)\}$. Furthermore, $\mathit{Trust}{=}\{(\mathtt{P1}, \mathit{less}, \mathtt{P2}), (\mathtt{P2}, \mathit{same}, \mathtt{P3}), (\mathtt{P4}, \mathit{less}, \mathtt{P2}), (\mathtt{P4}, \mathit{less}, \mathtt{P3})\}$, and Σ contains:

$$\begin{array}{rcll}
\Sigma(\mathtt{P1},\mathtt{P2}) & = & \{\forall xy(R^2(x,y) \rightarrow R^1(x,y))\}, & (3)\\
\Sigma(\mathtt{P2},\mathtt{P2}) & = & \{\forall x\forall y(R^2(x,y) \wedge S^2(x,y) \rightarrow \mathbf{false}), & (4)\\
 & & \quad \forall x\forall y\forall z(R^2(x,y) \wedge R^2(x,z) \rightarrow y = z)\}, & \\
\Sigma(\mathtt{P2},\mathtt{P3}) & = & \{\forall xy(R^2(x,y) \wedge R^3(x,y) \rightarrow \mathbf{false})\}, & (5)\\
\Sigma(\mathtt{P4},\mathtt{P2}) & = & \{\forall xyz(R^2(x,y) \wedge S^2(y,z) \rightarrow R^4(x,y,z))\}, & (6)\\
\Sigma(\mathtt{P4},\mathtt{P3}) & = & \{\forall xy(R^3(x,y) \rightarrow \exists z R^4(x,y,z))\}\cdot & (7)
\end{array}$$

The built-in atom **false** in (4) and (5) is false in every instance. The DECs in $\Sigma(\texttt{P2},\texttt{P2})$ are the local integrity constraints for P2, in this case a denial constraint (i.e. a prohibited join of positive atoms) and a functional dependency, requiring that (the values for) the first attribute functionally determines (determine the values for) the second one. Here, $\Sigma(\texttt{P4}) = \{\forall xyz(R^2(x,y) \wedge S^2(y,z) \rightarrow R^4(x,y,z)),$ $\forall xy(R^3(x,y) \rightarrow \exists z R^4(x,y,z))\}$.

The DEC in (7) is an RDEC, and all the others are UDECs. Some of the accessibility graphs are shown in Figure 3. Notice that $\mathcal{AC}(\texttt{P1}) = \{\texttt{P1},\texttt{P2},\texttt{P3}\}$, $\mathcal{AC}(\texttt{P2}) = \{\texttt{P2},\texttt{P3}\}$, $\mathcal{AC}(\texttt{P3}) = \{\texttt{P3}\}$, $\mathcal{AC}(\texttt{P4}) = \{\texttt{P2},\texttt{P3},\texttt{P4}\}$, $\mathcal{N}(\texttt{P1}) = \{\texttt{P1},\texttt{P2}\}$, $\mathcal{N}(\texttt{P2}) = \{\texttt{P2},\texttt{P3}\}$, $\mathcal{N}(\texttt{P3}) = \{\texttt{P3}\}$, $\mathcal{N}(\texttt{P4}) = \{\texttt{P2},\texttt{P3},\texttt{P4}\}$, $\mathcal{N}^\circ(\texttt{P1}) = \{\texttt{P2}\}$, $\mathcal{N}^\circ(\texttt{P2}) = \{\texttt{P3}\}$, $\mathcal{N}^\circ(\texttt{P3}) = \emptyset$, and $\mathcal{N}^\circ(\texttt{P4}) = \{\texttt{P2},\texttt{P3}\}$.

As an example, peer P4 only knows its schema and database, the DECs from it to peers P2 and P3, and how much it trusts them. More precisely, P4 knows its schema $\mathfrak{P}(\texttt{P4}) = \langle \mathcal{S}(\texttt{P4}), \Sigma(\texttt{P4}), \mathit{Trust}(\texttt{P4})\rangle$, with $\Sigma(\texttt{P4}) = \{\forall xyz(R^2(x,y) \wedge S^2(y,z) \rightarrow R^4(x,y,z)), \forall xy(R^3(x,y) \rightarrow \exists z R^4(x,y,z))\}$, and $\mathit{Trust}(\texttt{P4}) = \{(\texttt{P4}, \mathit{less}, \texttt{P2}), (\texttt{P4}, \mathit{less}, \texttt{P3})\}$. This peer also has its database instance $D(\texttt{P4})$.

The schema of P1 is $\mathfrak{P}(\texttt{P1}) = \langle \{R^1(\cdot,\cdot)\}, \{\forall xy(R^2(x,y) \rightarrow R^1(x,y))\}, \{(\texttt{P1}, \mathit{less}, \texttt{P2})\}\rangle$. As will be determined by the semantics later on, the combination of P1's DEC in (3) and the trust relationship to P2, will make P1 import all the missing data from the extension of its neighbor's R^2 into the extension of its own R^1. □

Example 2.2
(example 1.1 cont.) The PDES can be formalized through the schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$, where $\mathcal{P} = \{\texttt{P1},\texttt{P2}\}$, $\mathfrak{S} = \{\mathcal{S}(\texttt{P1}), \mathcal{S}(\texttt{P2})\}$, $\Sigma = \{\Sigma(\texttt{P1},\texttt{P2})\}$, $\Sigma(\texttt{P1},\texttt{P2}) = \{\forall xy(R^2(x,y) \wedge S^2(y,z) \rightarrow R^1(x,y,z)), \forall x(S^1(x) \rightarrow S^2(5,x))\}$, and $\mathit{Trust} = \{(\texttt{P1}, \mathit{less}, \texttt{P2})\}$.

The schema of P1 is $\mathfrak{P}(\texttt{P1}) = \langle \mathcal{S}(\texttt{P1}), \Sigma(\texttt{P1},\texttt{P2}), \{(\texttt{P1}, \mathit{less}, \texttt{P2})\}\rangle$, and that of P2 is $\mathfrak{P}(\texttt{P2}) = \langle \mathcal{S}(\texttt{P2}), \emptyset, \emptyset\rangle$. The instance for the PDES, $\mathfrak{D} = \{D(\texttt{P1}), D(\texttt{P2})\}$, contains the two instances shown in Figure 1. □

Remark 2.1
From now on we assume that the graph $\mathcal{G}(\mathfrak{P})$ is acyclic. As a consequence, for each particular peer P, $\mathcal{G}(\texttt{P})$ is also acyclic. This is a global assumption, not local to any particular peer, that will be used to define the semantics of a PDES. (A discussion around this and related assumptions can be found in the Online Appendix A.) □

Notice that according to its definition, the accessibility graph has no self-loops. The finiteness of the set of peers and the acyclicity of the accessibility graph imply that there must be "sink" (or terminal) peers, without any outgoing edges. Also notice that we are not making the assumption that for a peer P its initial instance $D(\texttt{P})$ has to satisfy the local constraints in $\Sigma(\texttt{P},\texttt{P})$. This applies in particular to sink peers in the accessibility graph. They can have local ICs, which may be violated.

3 A General Semantic Framework for a PDES

Before presenting the formal semantics of PDESs, we describe the intended semantics in intuitive and operational terms.

A query $\mathcal{Q} \in \mathcal{L}(\mathtt{P})$ is posed to a peer P by a certain user U, who may be an external user or another peer in $\mathcal{P}$. Now P, depending on the query predicates, inspects the DECs in $\Sigma(\mathtt{P})$ to identify data owned by other neighboring peers Q that may be related to its own data. If predicate $R^{\mathtt{Q}} \in \mathcal{S}(\mathtt{Q})$ appears in $\Sigma(\mathtt{P})$, then P requests from Q the (contents of the) relation $R^{\mathtt{Q}}$. Q returns to P a possible modified version of its relation, because Q may have to take into account data from its own neighbors (in the same way P did when it received the user query), which could produce local inconsistencies at Q's level, and they have to be repaired; and so on.

For the purpose of presenting the semantics of the system, that should be query-independent, we will assume for the moment that P receives whole relation instances from its neighbors. We emphasize that, in their turn, P's neighboring peers Q consider data from their own neighbors, and so on; data that may also be eventually used by P, by transitivity (cf. details below).

The data P receives from another peer Q is defined in a recursive manner, because Q may have connections to other peers who may contribute with data of their own. Eventually, after receiving the requested relation instances from its neighbors, P has now a database instance D for the expanded database schema $\mathcal{S}(\mathcal{N}(\mathtt{P}))$, which extends the initial instance $D(\mathtt{P})$ by adding predicates and data from its neighbors. D can be used to interpret the DECs in $\Sigma(\mathtt{P})$. Most likely, D will not satisfy $\Sigma(\mathtt{P})$.

When this extended instance D is inconsistent with respect to $\Sigma(\mathtt{P})$, different alternatives to restore consistency with respect to $\Sigma(\mathtt{P})$ can be considered, but the (possibly virtual) updates performed on the extended instance D will have to both respect the trust relationships and make sure that the consistent alternative instances stay *as close as possible* to instance D (for which a form of distance has to be introduced, as we do later in this section). In this way, a collection of consistent, and possibly virtual, extended instances for P's neighborhood emerges, the *neighborhood instances*. By restricting those instances to the schema of peer P, the *solution instances* for P are obtained.

Now, we give the precise definition of neighborhood solution. In order to do so, we will assume that it is possible to compare arbitrary instances D_1, D_2 with respect to a fixed instance D by means of a preorder relation $\preceq_D$ (i.e. a reflexive and transitive binary relation). If D, D_1, D_2 are database instances for the same schema, the intuition behind the relationship $D_1 \preceq_D D_2$ is that D_1 *is at least as close to* D *as* D_2 *(is to* D*)*. We can define $D_1 \prec_D D_2$ iff $D_1 \preceq_D D_2$, but not $D_2 \preceq_D D_1$, with the intuition that D_1 is closer to D than D_2.

In our semantics, the participating instances will always be for a neighborhood schema $\mathcal{S}(\mathcal{N}(\mathtt{P}))$ of a peer P. Even more, the preorder relation will also depend on $\Sigma(\mathtt{P})$ (which has to be satisfied).

As a consequence, we will assume, for D *a fixed instance for* $\mathcal{S}(\mathcal{N}(\mathtt{P}))$*, the existence of a preorder relation* $\preceq_D^{\Sigma(P)}$ *on instances for the schema* $\mathcal{S}(\mathcal{N}(\mathtt{P}))$*.*

Definition 3.1

Given a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust} \rangle$, a peer $\mathtt{P} \in \mathcal{P}$, and an instance $\bar{D}$ for the schema $\mathcal{S}(\mathcal{N}(\mathtt{P}))$:

(*a*) An instance D' for the schema $\mathcal{S}(\mathcal{N}(\mathtt{P}))$ is a *neighborhood solution* for $\mathtt{P}$ and $\bar{D}$ if:

  (*i*) $D' \models \Sigma(\mathtt{P})$.

  (*ii*) There is no instance D'' satisfying *(i)*, and $D'' \prec_{\bar{D}}^{\Sigma(\mathtt{P})} D'$.

  (*iii*) $D'\!\downharpoonright\!\{R\} = \bar{D}\!\downharpoonright\!\{R\}$ for every predicate $R \in \mathcal{S}(\mathtt{Q})$ with $(\mathtt{P}, \mathit{less}, \mathtt{Q}) \in \mathit{Trust}$.

(*b*) $\mathit{NS}^{\mathfrak{P}}(\mathtt{P}, \bar{D})$ denotes the set of neighborhood solutions for $\mathtt{P}$ and $\bar{D}$.
When clear from the context, we will simply use $\mathit{NS}(\mathtt{P}, \bar{D})$. □

A neighborhood solution for P is a database for its whole neighborhood that satisfies P's DECs (including its local constraints), and respects its trust relationships. A neighborhood solution also stays as close as possible to the original neighborhood instance, while staying the same for trustable peers. In operational and intuitive terms, a minimal data set with respect to Definition 3.1 is imported or given up to satisfy the DECs.

Notice that Definition 3.1 introduces an abstract *repair semantics*, similar to those used to handle inconsistency in single databases (Bertossi 2011). For example, a well-studied repair semantics for single databases was introduced in (Arenas et al. 1999). It is based on insertions or deletions of whole tuples into/from the original inconsistent instance D, and the "distances" of two instances D_1, D_2 from D are compared using the symmetric differences: $D_1 \preceq_D^{\Delta,\Sigma(\mathtt{P})} D_2 :\Leftrightarrow \Delta(D, D_1) \subseteq \Delta(D, D_2)$.[5]

For a peer P in isolation, i.e. with $\mathcal{N}^{\circ}(\mathtt{P}) = \emptyset$, a neighborhood solution of its original instance $D(\mathtt{P})$ will simply be an instance D' for its schema $\mathcal{S}(\mathtt{P})$, such that: (a) $D' \models \Sigma(\mathtt{P}, \mathtt{P})$ (i.e. P's own integrity constraints); and (b) There is no instance D'' satisfying (a) and $D'' \prec_{D(\mathtt{P})}^{\Sigma(\mathtt{P},\mathtt{P})} D'$. This defines a class of *repairs* of instance $D(\mathtt{P})$ with respect to $\Sigma(\mathtt{P}, \mathtt{P})$.[6] Thus, this neighborhood-solution semantics for PDESs naturally extends the notion of repair to neighborhood solutions.

Example 3.1
(examples 1.1 and 2.2 cont.) The elements of the PDES schema are all as before, in particular,

$$\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall xy(R^2(x, y) \wedge S^2(y, z) \rightarrow R^1(x, y, z)),\ \forall x(S^1(x) \rightarrow S^2(5, x))\}, \quad (8)$$

but this time consider $\{(\mathtt{P1}, \mathit{same}, \mathtt{P2})\} \in \mathit{Trust}$.

Now assume, in this example, for illustration purposes, that the preorder relations are defined by $D_1 \preceq_D^{\Delta,\Sigma(\mathtt{P})} D_2$ iff $\Delta(D, D_1) \subseteq \Delta(D, D_2)$.

If a query is posed to P1, it will have to compute (possibly virtually) its neighborhood solutions. Peer P1's knowledge of the schema is limited to its own schema $\mathfrak{P}(\mathtt{P1}) = \langle \mathcal{S}(\mathtt{P1}), \Sigma(\mathtt{P1}), \mathit{Trust}(\mathtt{P1})\rangle$ where $\mathcal{S}(\mathtt{P1}) = \{R^1(\cdot,\cdot,\cdot), S^1(\cdot)\}$, $\Sigma(\mathtt{P1}) = \Sigma(\mathtt{P1}, \mathtt{P2})$, and $\mathit{Trust}(\mathtt{P1}) = \{(\mathtt{P1}, \mathit{same}, \mathtt{P2})\}$.

In order to enforce $\Sigma(\mathtt{P1})$, P1 poses two atomic queries to peer P2: $\mathcal{Q}_1(x, y)$: $R^2(x, y)$ and $\mathcal{Q}_2(x, y)$: $S^2(x, y)$. Since P2 is not related to any other peer, and has no local ICs, it will provide as answers the content of those relations in $D(\mathtt{P2})$. In

[5] For sets S_1 and S_2, $\Delta(S_1, S_2) := (S_1 \smallsetminus S_2) \cup (S_2 \smallsetminus S_1)$.
[6] We are using abstract preorder relations to define repair semantics. It should also be possible to use abstract "distance measures" to define repairs and repair semantics. Cf. (Arieli et al. 2003).

R^1			S^1	R^2		S^2	
c	4	2	3	c	4	4	2
f	3	5	7	d	5	5	3

Fig. 4. A database instance D for schema $\mathcal{S}(\mathcal{N}(\texttt{P1}))$

R^1			S^1	R^2		S^2	
c	4	2		c	4	4	2
f	3	5		d	5		

R^1			S^1	R^2		S^2	
c	4	2	7	c	4	4	2
f	3	5		d	5	5	7
d	5	7					

R^1			S^1	R^2		S^2	
c	4	2	3	c	4	4	2
f	3	5	7			5	3
						5	7

R^1			S^1	R^2		S^2	
c	4	2	3	c	4	4	2
f	3	5				5	3

R^1			S^1	R^2		S^2	
c	4	2	3	c	4	4	2
f	3	5		d	5	5	3
d	5	3					

R^1			S^1	R^2		S^2	
c	4	2	3	c	4	4	2
f	3	5	7	d	5	5	3
d	5	3				5	7

Fig. 5. Several neighborhood solution instances for peer P1

this way, P1 has an extended instance D, shown in Figure 4, that corresponds to the union of the two instances in Figure 1, that is, $D = D(\texttt{P1}) \cup D(\texttt{P2})$. Since D does not satisfy $\Sigma(\texttt{P1})$, D has to be repaired.

Since $(\texttt{P1}, \mathit{same}, \texttt{P2})$, there are several neighborhood solutions for P1, which are obtained by virtually modifying both peers' data. For example, the inconsistencies with respect to the first UDEC in (8) can be (minimally) solved by either removing $R^2(d,5)$ from R^2, or removing $S^2(5,3)$ from S^2, or inserting $R^1(d,5,3)$ into R^1. If $S^2(5,3)$ were removed from S^2, a new inconsistency is created with respect to the second UDEC in (8). This one can be solved by removing $S^1(3)$ from S^1. The inconsistencies with respect to the second DEC can be solved by either removing $S^1(7)$ from S^1 or inserting $S^2(5,7)$ into S^2.

Figure 5 shows the six neighborhood solutions in $\mathit{NS}^{\mathfrak{P}}(\texttt{P1}, D)$. All these neighborhood instances are *repairs* in the sense of (Arenas et al. 1999) of instance D with respect to P1's DECs. If we are locally interested only in P1, we consider their restrictions to P1' schema $\mathcal{S}(\texttt{P1})$. □

As the following example shows, trust relationships in combination with inconsistent DECs may cause that neighborhood solutions do not exist.

Example 3.2
Consider a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust} \rangle$ with $\mathcal{P} = \{\texttt{P}, \texttt{Q}, \texttt{R}\}$, and

1. $\mathfrak{S} = \{\mathcal{S}(\texttt{P}), \mathcal{S}(\texttt{Q}), \mathcal{S}(\texttt{R})\}$, $\mathcal{S}(\texttt{P}) = \{P\}$, $\mathcal{S}(\texttt{Q}) = \{Q\}$, $\mathcal{S}(\texttt{R}) = \{R\}$.

2. $\Sigma = \{\Sigma(\mathtt{P},\mathtt{Q}), \Sigma(\mathtt{P},\mathtt{R}))\}$, $\Sigma(\mathtt{P},\mathtt{Q}) = \{\forall xy(Q(x,y) \rightarrow P(x,y))\}$,
$\Sigma(\mathtt{P},\mathtt{R}) = \{\forall xy(P(x,y) \rightarrow R(x,y))\}$.
3. $\mathit{Trust} = \{(\mathtt{P}, \mathit{less}, \mathtt{Q}),\ (\mathtt{P}, \mathit{less}, \mathtt{R})\}$.
4. $\mathfrak{D} = \{D(\mathtt{P}), D(\mathtt{Q}), D(\mathtt{R})\}$, with $D(\mathtt{P}) = D(\mathtt{R}) = \emptyset$, and $D(\mathtt{Q}) = \{Q(a,b)\}$.

For peer P and $D = D(\mathtt{P}) \cup D(\mathtt{Q}) \cup D(\mathtt{R}) = \{Q(a,b)\}$, there is no neighborhood solution, i.e. $\mathit{NS}(\mathtt{P}, D(\mathtt{P}) \cup D(\mathtt{Q}) \cup D(\mathtt{R}))$ is empty. This is because P's DEC with Q forces the insertion of $P(a,b)$, but at the same time the interaction of P with R requires relation P to be empty. Since peer P trusts both peers more than itself, there is no neighborhood solution that satisfies the DECs and respect the trust relations.

Notice that the DEC in $\Sigma(\mathtt{P},\mathtt{R})$ acts as a restriction on tuples for its owner, P, rather than as a tuple generator for it. □

In the previous example, the trust relationships impose unsolvable conditions on neighborhood solutions. However, if all the trust relationships are of the flexible form $(\mathtt{P}, \mathit{same}, \mathtt{Q})$, a peer always has solutions.

Proposition 3.1
Consider a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$ and a peer $\mathtt{P} \in \mathcal{P}$ whose trust relationships are all of the form $(\mathtt{P}, \mathit{same}, \mathtt{Q})$. Given an instance $\bar{D}$ for $\mathcal{S}(\mathcal{N}(\mathtt{P}))$, peer P always has a neighborhood solution for $\bar{D}$. □

Having defined the notion of neighborhood solution for a peer, now we can define the notion of solution for a peer, which is a local instance but takes all the accessible peers into account. The data distributed across different peers has to be appropriately gathered to build solution instances for the peer. This definition of solution instance is recursive, and appeals to that of neighborhood solution. The intuition and idea are as follows: (a) Assume each neighbor Q of peer P has the class $\mathit{Sol}(\mathtt{Q}, \mathfrak{D})$ of its solution instances (hence the recursion). Each of these Q passes to P the intersection $\bigcap \mathit{Sol}(\mathtt{Q}, \mathfrak{D})$ of its solution set.[7] With them and its own initial instance $D(\mathtt{P})$, P builds an instance D for its neighborhood $\mathcal{N}(\mathtt{P})$. The solutions for P become the restrictions to P's schema of the neighborhood solutions for P with respect to D, $\Sigma(\mathtt{P})$, and P's trust relationships.

Under this recursive definition, the solutions for the neighbors have to be determined, under the same semantics. Base cases of the recursion are peers with no DECs or with only local constraints, that is, when either $\Sigma(\mathtt{P}) = \emptyset$ or $\Sigma(\mathtt{P}) = \{\Sigma(\mathtt{P},\mathtt{P})\}$. These peers are sinks in the accessibility graph of P. In this regard, we recall that we made the assumption that the accessibility graphs are acyclic.

Definition 3.2
Given an instance $\mathfrak{D}$ for the PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$, and a peer P, an instance D for the schema $\mathcal{S}(\mathtt{P})$ is a *solution instance* (or simply *solution*) for P, denoted $D \in \mathit{Sol}^{\mathfrak{P}}(\mathtt{P}, \mathfrak{D})$ (or simply, $\mathit{Sol}(\mathtt{P}, \mathfrak{D})$), iff:

[7] When $\mathcal{C}$ is a class of sets, then we will usually denote with $\bigcap \mathcal{C}$ the intersection of its elements, i.e. $\bigcap \mathcal{C} := \bigcap_{C \in \mathcal{C}} C$.

(a) For $\Sigma(\mathtt{P}) = \emptyset$: $D = D(\mathtt{P})$ ($\mathtt{P}$'s initial instance, the projection of $\mathfrak{D}$ on $\mathcal{S}(\mathtt{P})$).
(b) For $\Sigma(\mathtt{P}) = \{\Sigma(\mathtt{P},\mathtt{P})\}$: $D \in \mathit{NS}^{\mathfrak{P}}(\mathtt{P}, D(\mathtt{P}))$.
(c) For $\emptyset \neq \Sigma(\mathtt{P}) \neq \{\Sigma(\mathtt{P},\mathtt{P})\}$: $D = \overline{D} \downharpoonright \mathcal{S}(\mathtt{P})$, where $\overline{D} \in \mathit{NS}^{\mathfrak{P}}(\mathtt{P}, D(\mathtt{P}) \cup \bigcup_{\mathtt{Q} \in \mathcal{N}^{\circ}(\mathtt{P})} \mathit{Core}(\mathtt{Q}, \mathfrak{D}))$.
Here, $\mathit{Core}(\mathtt{Q}, \mathfrak{D}) := \bigcap \mathit{Sol}^{\mathfrak{P}}(\mathtt{Q}, \mathfrak{D})$ when $\mathit{Sol}^{\mathfrak{P}}(\mathtt{Q}, \mathfrak{D}) \neq \emptyset$; and $\mathit{Core}(\mathtt{Q}, \mathfrak{D}) := \{\mathbf{inc}_{\mathtt{Q}}\}$, otherwise. The propositional built-in predicate $\mathbf{inc}_{\mathtt{Q}}$ in $\mathcal{S}(\mathtt{Q})$ is true of an instance iff it is contained in the latter as an atom. □

The base cases of the recursion are (a) and (b), where the solutions of a peer can be computed without data from other peers. Case (c) corresponds to the properly recursive case, where, before determining $\mathtt{P}$'s solutions, there is an extended instance $\overline{D}$ around $\mathtt{P}$ formed by its local instance $D(\mathtt{P})$ plus, for each neighbor $\mathtt{Q}$, the intersection, $\mathit{Core}(\mathtt{Q}, \mathfrak{D})$, of all its solutions. The combined database $\overline{D}$ is for the schema $\mathcal{S}(\mathcal{N}(\mathtt{P}))$, and starting from $\overline{D}$, neighborhood solutions for $\mathtt{P}$ can be determined; and their restrictions to $\mathtt{P}$'s schema become $\mathtt{P}$'s solutions.

Although, in Definition 3.2, cases (a) and (b) are special cases of (c), we include them explicitly, for clarity. Notice that case (b) amounts to obtaining the repairs of a local instance with respect to a set of local integrity constraints.

The following is a immediate consequence of Proposition 3.1.

Corollary 3.1
Consider a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust} \rangle$ with trust relationships all of the form $(\mathtt{R}, \mathit{same}, \mathtt{Q})$, and a peer $\mathtt{P} \in \mathcal{P}$. Given an instance $\mathfrak{D}$ for $\mathfrak{P}$, peer $\mathtt{P}$ always has a solution, that is, $\mathit{Sol}^{\mathfrak{P}}(\mathtt{P}, \mathfrak{D}) \neq \emptyset$. □

Remark 3.1
As we have seen, even in the absence of cycles in $\mathcal{G}(\mathcal{P})$, a peer may have no neighborhood solutions (cf. Example 3.2), and then no solutions, i.e. $\mathit{Sol}(\mathtt{P}, \mathfrak{D}) = \emptyset$. That is why, in this case, we make the convention in Definition 3.2 that $\mathit{Core}(\mathtt{P}, \mathfrak{D}) = \{\mathbf{inc}_{\mathtt{P}}\}$. When $\mathit{Sol}(\mathtt{P}, \mathfrak{D}) = \emptyset$, if another peer requests from $\mathtt{P}$ the intersection of its solutions, $\mathtt{P}$ returns the instance $\{\mathbf{inc}_{\mathtt{P}}\}$.

The idea is that when a peer becomes inconsistent, it makes another peer $\mathtt{Q}$ who requests its data notice its inconsistency by sending this special instance. In this case, $\mathtt{Q}$ is expected to ignore its mappings with $\mathtt{P}$. This can be made precise and formally accommodated by modifying the DECs through the introduction of an extra conjunct in the antecedent. For example, the DEC $\Sigma(\mathtt{Q}, \mathtt{P}) = \{\forall x(P(x) \rightarrow Q(x))\}$ would be replaced by $\{\forall x(P(x) \wedge \neg \mathbf{inc}_{\mathtt{P}} \rightarrow Q(x))\}$, which would become trivially satisfied. Alternatively, if we want to keep the built-ins in the consequent, we could replace $\Sigma(\mathtt{Q}, \mathtt{P})$ by $\{\forall x(P(x) \rightarrow (Q(x) \vee \mathbf{inc}_{\mathtt{P}}))\}$.

In order not to complicate the notation, we will refrain from explicitly introducing the $\mathbf{inc}_{\mathtt{P}}$s in the DECs. □

According to this convention about the treatment of peers without solutions, a peer that becomes intrinsically inconsistent, "irreparable", is ignored by its neighbors when they receive the notification of inconsistency. However, this form of ignoring is put here on a solid logical foot, that is compatible and uniform with the treatment of other peers. As long as a peer declares itself as inconsistent, there is no much a neighbor can do. However, an inconsistent peer might decide to relax its own consistency requirements and send to other peers only "partially consistent" data, which would be transparent to those receiving peers.[8]

The peer consistent answers from a peer to a query are the semantically correct answers, which means that when answering the query, the peer consistently considers the data of its neighbors and the trust relationships with them.

Definition 3.3
Consider an instance $\mathfrak{D}$ for the PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$, and a peer $\mathtt{P} \in \mathcal{P}$. Let $\mathcal{Q}(\bar{x}) \in \mathcal{L}(\mathtt{P})$ be a query, with $\bar{x}$ a possibly empty list of free variables.

1. If $\mathit{Sol}(\mathtt{P}, \mathfrak{D}) \neq \emptyset$:
   (a) If $\bar{x} \neq \emptyset$, a finite sequence $\bar{c}$ of constants in $\mathcal{U}$ of the same length as $\bar{x}$ is a *peer consistent answer* (PCA) to $\mathcal{Q}$ from $\mathtt{P}$ iff $D \models \mathcal{Q}[\bar{c}]$ for every $D \in \mathit{Sol}(\mathtt{P}, \mathfrak{D})$.
   (b) If $\mathcal{Q}$ is Boolean and $D \models \mathcal{Q}$ for every $D \in \mathit{Sol}(\mathtt{P}, \mathfrak{D})$, then *yes* is the only PCA to $\mathcal{Q}$.
2. If $\mathit{Sol}(\mathtt{P}, \mathfrak{D}) = \emptyset$, then $\mathbf{inc}_{\mathtt{P}}$ is the only PCA to $\mathcal{Q}(\bar{x})$. □

For illustration, in Example 3.1, if P1 has to answer a query $\mathcal{Q} \in \mathcal{L}(\mathtt{P1})$, it returns the answers that are simultaneously true in all its neighborhood solutions. For example, under this semantics, the answers to query $\mathcal{Q}(x) : \exists yz R^1(x, y, z)$ posed to P1 will be $\langle c\rangle, \langle f\rangle$, which are shared by all the six neighborhood solutions for P1 (or better, by their restrictions to P1's schema). The answers to queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ posed by P1 to P2 are answered in the same way, taking into consideration the P2's DECs.

We can see that the answers from a peer to a query are *certain answers* (Imielinski and Lipski 1984). This condition makes the data moved from one peer to a neighbor always certain. In particular, this allows us to treat external queries and inter-peer queries in a uniform manner. In particular, we will be in position to conceive and implement the passage of data from one peer to a neighbor as a query answering process.

Example 3.3
Consider the PDES schema $\mathfrak{P}$ and instance $\mathfrak{D}$ represented in graph $\mathcal{G}(\mathfrak{P})$ in Figure 6. Here, $\Sigma = \{\Sigma(\mathtt{P1}, \mathtt{P2}), \Sigma(\mathtt{P2}, \mathtt{P3}), \Sigma(\mathtt{P4}, \mathtt{P3})\}$, and:
- $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall xyz\ (R^2(x, y) \wedge S^2(y, z) \rightarrow R^1(x, y, z)), \forall x\ (S^1(x) \rightarrow S^2(5, x))\}$.
- $\Sigma(\mathtt{P2}, \mathtt{P3}) = \{\forall xy\ (S^2(x, y) \rightarrow R^3(x, y))\}$.
- $\Sigma(\mathtt{P4}, \mathtt{P3}) = \{\forall xyz\ (R^3(x, y) \rightarrow R^4(x, y, 3))\}$.

In intuitive and procedural terms, 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 $\Sigma(\mathtt{P2},$

[8] An alternative to this design choice could be, in the case a peer P trusts an inconsistent peer Q more than itself, that P becomes or declares itself inconsistent as well. This alternative may be worth exploring, but we do not pursue it here any further.

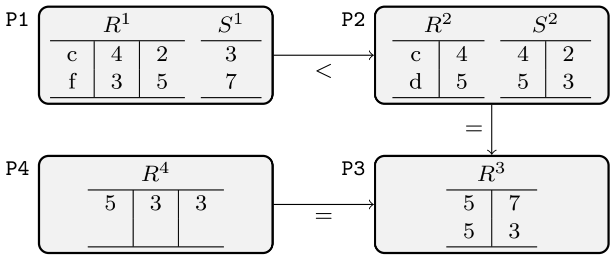


Fig. 6. PDES for Example 3.3

P3). Since P3 is not connected to any other peer, it will answer P2's queries using its initial material instance $D(\texttt{P3})$. Thus, the solutions for P1 and its peer consistent answers will be affected by the peers in $\mathcal{AC}(\texttt{P1}) = \{\texttt{P1}, \texttt{P2}, \texttt{P3}\}$. More precisely, we can now apply the definitions of solution instance and peer consistent answer.

Since P3 has neither DECs with other peers nor local integrity constraints, its only neighborhood solution is its instance $D(\texttt{P3}) \in \mathfrak{D}$, which is sent back to P2. Now, P2 has to repair the extended instance $D = \{R^2(c, 4),\ R^2(d, 5),\ S^2(4, 2),\ S^2(5, 3),\ R^3(5, 7),\ R^3(5, 3)\}$ with respect to $\Sigma(\texttt{P2}, \texttt{P3})$, which is not satisfied due to the presence of tuple $S^2(4, 2)$. As P2 trusts P3 the same as itself, it can modify its own data or the data it got from P3.

Assuming -for illustration purposes for the moment- that the preorder relation on instances is given in terms of the symmetric set-difference, 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 $\{R^2(c, 4),\ R^2(d, 5),\ S^2(4, 2),\ S^2(5, 3),\ R^3(5, 7),\ R^3(5, 3),\ R^3(4, 2)\}$. They minimally depart from D, and their restrictions to P2's schema lead to two solutions for P2: $\mathit{Sol}(\texttt{P2}, \mathfrak{D}) = \{\{R^2(c, 4), R^2(d, 5), S^2(5, 3)\}, \{R^2(c, 4),\ R^2(d, 5),\ S^2(4, 2), S^2(5, 3)\}\}$.

Peer P2 will send to P1 the intersection of its solutions, namely $\mathit{Core}(\texttt{P2}, \mathfrak{D}) = \bigcap \mathit{Sol}(\texttt{P2}, \mathfrak{D}) = \{R^2(c, 4),\ R^2(d, 5),\ S^2(5, 3)\}$. Now, P1 has to repair the extended instance $\{R^1(c, 4, 2),\ R^1(f, 3, 5),\ S^1(3),\ S^1(7),\ R^2(c, 4),\ R^2(d, 5),\ S^2(5, 3)\}$ with respect to $\Sigma(\texttt{P1}, \texttt{P2})$, which is not satisfied.

Since P1 trusts P2 more, it will solve inconsistencies by modifying its own data, in this case inserting tuples into R^1 and deleting tuples from S^1. This produces 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, $\mathit{Sol}(\texttt{P1}, \mathfrak{D}) = \{\{R^1(c, 4, 2), R^1(f, 3, 5), R^1(d, 5, 3), S^1(3)\}\}$.

If P1 had received the query $\mathcal{Q}(x) : \exists yz(R^1(x, y, z) \wedge S^1(y))$, the only peer consistent answer would have been $\langle f \rangle$, obtained from its only solution. □

Notice that when a peer Q passes its core, $\mathit{Core}(\texttt{Q}, \bar{D})$, with $\bar{D}$ a neighborhood instance, to a peer P, it is delivering the peers consistent answers to the atomic queries from P to Q of the form $\mathcal{Q}^R(\bar{x})\colon R(\bar{x})$, where $R \in \mathcal{S}(\texttt{Q})$. However, when P computes its local PCAs to a query, it does not use its own core, $\bigcap \mathit{Sol}(\texttt{P}, \mathfrak{D})$, but the collection $\mathit{Sol}(\texttt{P}, \mathfrak{D})$ as a whole. The reason is that this core is unnecessarily restrictive for peer consistent query answering. For example, if the query is $\mathcal{Q} : \exists x P(x)$, and P's solutions are $\{P(a)\}$ and $\{P(b)\}$, the PCA to this Boolean query would be *no* if evaluated on the empty core, but *yes* according to our definition.

Remark 3.2
Notice that the definitions of solution instance for a peer and of peer consistent answer are parameterized by:
(a) The class of data exchange constraints. We have considered certain syntactic classes of FO-sentences, and we will also do so in the rest of this work. However, our presentation so far has been general enough to accommodate broader classes of FO sentences in the combined language of any two peers. The restriction on the classes of DECs has not been required or used yet.

(b) A notion of satisfaction, $D \models \varphi$, where D is a database instance and φ is a FO sentence in the language of D's schema. Most commonly, D is a relational database considered as a FO structure, and classical logical satisfaction is used. However, if D contains uncertain information, then we may have to depart from FO logic, as we will see in Section 4, for a particular PDES semantics.

(c) The preorder relations $\preceq_D^{\Sigma(\mathtt{P})}$ between instances for the schema of an instance D. Different preorder relations can be considered. In the examples above we have considered, just to fix ideas, the common preorder based on the symmetric set difference between instances.

(d) The repair semantics, i.e. by a characterization of the instances that minimally depart from a given one D in order to satisfy local ICs or DECs between two peers. The repair semantics is based on the preorders $D'' \preceq_D^{\Sigma(\mathtt{P})} D'$, and the associated minimality conditions (what we did in Definition 3.1). □

4 Towards a Special PDES Semantics with NULL

Considering the rather abstract nature and flexibility of the semantic framework introduced in Section 3, in this section, we make specific commitments about the semantic parameters discussed in Remark 3.2.

In the rest of this section, we consider a classical relational schema $\Sigma = (\mathcal{U}, \mathcal{R}, \mathcal{B})$, consisting of the data domain, a set of database predicates, and a set of built-in predicates.[9] Now, we will introduce in the schema some extra elements related to the special constant, *null*, that we will use in the rest of this work.

Remark 4.1
We assume from now on that every attribute domain, $Adom(A)$, contains the constant *null*. In particular, $null \in \mathcal{U}$. Furthermore, among the built-in predicates in $\mathcal{B}$, we will also find $IsNull(\cdot)$, and $IsNotNull(\cdot)$. The first one is true only with constant $null$, and the second one, on any constant c other than $null$.[10] Accordingly, *null* and these built-in predicates may appear in DECs as defined in (1) and (2), in particular, in integrity constraints, and also in queries. They all become sentences or formulas of the FO language $\mathcal{L}(\Sigma^{\mathtt{null}})$ associated to $\Sigma^{\mathtt{null}} = (\mathcal{U}, \mathcal{R}, \mathcal{B}^{\mathtt{null}})$, with $\mathcal{B}^{\mathtt{null}} = \mathcal{B} \cup \{IsNull, IsNotNull\}$. □

[9] In the coming sections, where we will apply the semantics of this section, $\mathcal{R}$ will be the set of database predicates for a peer or a pair thereof.
[10] Each is the negation of the other, but we will keep both in order to avoid using explicit negation.

The special constant *null* in the data domain $\mathcal{U}$ is intended to behave and be used as the null value, NULL, in SQL relational databases. The new, unary, built-in predicates correspond to the SQL predicates `IS NULL` and `IS NOT NULL`, used to check null values. Constant *null* may appear in database tuples, and will be used to restore consistency with respect to DECs and ICs.

Using a single null, with its SQL semantics, is clearly different from using multiple labeled null values, as is done in data exchange for enforcing the satisfaction of existential quantifications (Kolaitis 2005). It is also different from using arbitrary elements of the underlying data domain for the same purpose (Cali et al. 2003). However, to a large extent, the semantics of this section could be developed without the specific restrictions imposed on the representation and use of null values, adopting other forms of handling incomplete and inconsistent data. Cf. Section 7 and the Online Appendix A for additional discussions of these issues.

The semantics of *null* we introduce next captures the way nulls are handled by relational DBMSs that follow the SQL standard. More precisely, our semantics provides a partial logical reconstruction in first-order predicate logic of the way nulls are handled in SQL databases. It refines and extend previous work presented in (Bravo and Bertossi 2006) on database repairs with and in the presence of `NULL`.

The SQL Standard leaves many issues around `NULL` unspecified, and different DBMSs depart from the standard in different ways. As a consequence, it is not possible to provide a full logical reconstruction of SQL databases with `NULL`. For this reason, Accordingly, our semantics concentrates on the notion of satisfaction of DECs and ICs, and query answering for a broad classes of queries. Furthermore, the proposed semantics extends the "classical" notion of DEC and IC satisfaction, and query answering in databases without `NULL`.[11]

The rest of this section continues as follows. In Section 4.1, we illustrate some of the elements of and issues around the notion of query answer as used in SQL databases with `NULL`. It serves a motivation for Section 4.2, where we formalize the semantics of query answering. In Section 4.3, we introduce a rewriting-based semantics for constraint satisfaction in the presence of `NULL`, leading to a fully classical semantics. Finally, in Section 4.4, we apply the rewriting methodology to the semantics of query answering under `NULL`.

4.1 Query answering under `NULL`*: motivation*

A tuple $\bar{c}$ of elements of $\mathcal{U}$ is an answer to query $\mathcal{Q}(\bar{x})$, denoted $D \models_N \mathcal{Q}(\bar{c})$, if the formula (that represents) $\mathcal{Q}$ is *classically true* when the quantifiers on its *relevant variables* (or attributes) run over $(\mathcal{U} \smallsetminus \{null\})$; and those on the non-relevant variables run over $\mathcal{U}$. The free relevant variables, i.e. relevant among those in $\bar{x}$, cannot take the value *null* either. (Relevance is made precise in Section 4.2.)

[11] We reserve the use of constant `NULL` to illustrate issues in relation its use in SQL databases; and to emphasize this, also in subsection titles. Otherwise, we keep using only the constant *null* of the underlying data domain.

Example 4.1
Consider the instance D_2 and the query below:

R	A	B	C
	1	1	1
	2	*null*	*null*
	null	3	3

S	B
	null
	1
	3

$$\mathcal{Q}_2(x)\colon\ \exists y \exists z (R(x,y,z) \wedge S(y) \wedge y > 2)\cdot \qquad (9)$$

A variable v (quantified or not) in a conjunctive query is *relevant* if it appears (non-trivially) twice in the formula after the quantifier prefix. Occurrences of the form $v = \mathit{null}$ and $v \neq \mathit{null}$ do not count though.

In query (9), the only relevant quantified variable is y, because it participates in a join and a built-in in the quantifier-free matrix of (9). So, there are two reasons for y to be relevant. The only free variable is x, which is not relevant. As for query answers, the only candidate values for x are: $\mathit{null}, 2, 1$. In this case, *null* is a candidate value because x is a non-relevant variable.

First, $x = \mathit{null}$ is an answer to the query, because the formula $\exists y \exists z (R(x,y,z) \wedge S(y) \wedge y > 2)$ is true in D_2, with a non-null witness value for y and a witness value for z that combined make the (non-quantified) formula true. Namely, $y = 3, z = 3$. So, it holds $D_2 \models_N \mathcal{Q}_2[\mathit{null}]$.

Next, $x = 2$ is not an answer. For this value of x, because the candidate value for y, namely *null* that accompanies 2 in P, makes the formula $(R(x,y,z) \wedge S(y) \wedge y > 2)$ false. Even if it were true, this value for y would not be allowed.

Finally, $x = 1$ is not an answer, because the only candidate value for y, namely 1, makes the formula false. In consequence, *null* is the only answer. □

The next example with SQL queries and `NULL` provides additional intuition and motivation for the formal semantics of Section 4.2. Notice the use in logical queries of the new unary predicates *IsNull* and *IsNotNull*.

Example 4.2
Consider the schema $\mathcal{S} = \{R(A,B), S(B,C)\}$ and the instance in the table below. In it `NULL` is the SQL null. If this instance is stored in an SQL database, we can observe the behavior of the following queries when they are directly translated into SQL and run on an SQL DB:

R	A	B
	a	b
	a	c
	d	NULL
	d	e
	u	u
	v	NULL
	v	r
	NULL	NULL

S	B	C
	b	h
	NULL	s
	l	m

(a) $\mathcal{Q}_1(x,y)\colon R(x,y) \ \wedge\ y = \mathit{null}$
SQL: `Select * from R where B = NULL;`
Result: No tuple

(b) $\mathcal{Q}_1'(x,y)\colon R(x,y) \ \wedge\ \mathit{IsNull}(y)$
SQL: Now uses `IS NULL`
Result: $\langle$d, `NULL`$\rangle$, $\langle$v, `NULL`$\rangle$, $\langle$`NULL`, `NULL`$\rangle$

(c) $\mathcal{Q}_2(x,y):\ R(x,y) \wedge y \neq \mathit{null}$
SQL: `Select * from R where B <> NULL;`
Result: No tuple

(d) $\mathcal{Q}_2'(x,y):\ R(x,y) \wedge \mathit{IsNotNull}(y)$
SQL: Now uses `IS NOT NULL`
Answer: The five expected tuples

(e) $\mathcal{Q}_3(x,y): R(x,y) \wedge x = y$
SQL: `Select * from R where A = B;`
Result: $\langle \mathrm{u},\mathrm{u}\rangle$

(f) $\mathcal{Q}_4(x,y): R(x,y) \wedge x \neq y$
SQL: `Select * from R where A <> B;`
Result: $\langle \mathrm{a},\mathrm{b}\rangle, \langle \mathrm{a},\mathrm{c}\rangle, \langle \mathrm{d},\mathrm{e}\rangle, \langle \mathrm{v},\mathrm{r}\rangle$

(g) $\mathcal{Q}_5(x,y,x,z): R(x,y) \wedge R(x,z) \wedge y \neq z$
SQL: `Select * from R r1, R r2 where r1.A = r2.A and r1.B <> r2.B;`
Result: $\langle \mathrm{a},\mathrm{b},\mathrm{a},\mathrm{c}\rangle, \langle \mathrm{a},\mathrm{c},\mathrm{a},\mathrm{b}\rangle$

(h) $\mathcal{Q}_6(x,y,z,t)$: $R(x,y) \wedge S(z,t) \wedge y = z$
SQL: `Select * from R r1, S s1 where r1.B = s1.B;`
Result: $\langle \mathrm{a},\mathrm{b},\mathrm{b},\mathrm{h}\rangle$

(i) As in (h), but now SQL: `Select * from R r1 join S s1 on r1.B = s1.B;`
Result:[12] $\langle \mathrm{a},\mathrm{b},\mathrm{b},\mathrm{h}\rangle$

(j) $\mathcal{Q}_7(x,y,z,t): R(x,y) \wedge S(z,t) \wedge y \neq z.$ In SQL:
`Select R1.A, R1.B, S1.B, S1.C from R R1, S S1 where R1.B <> S1.B';`
Result: $\langle \mathrm{a},\mathrm{c},\mathrm{b},\mathrm{h}\rangle, \langle \mathrm{d},\mathrm{e},\mathrm{b},\mathrm{h}\rangle, \langle \mathrm{u},\mathrm{u},\mathrm{b},\mathrm{h}\rangle, \langle \mathrm{v},\mathrm{r},\mathrm{b},\mathrm{h}\rangle$, $\langle \mathrm{a},\mathrm{b},\mathrm{l},\mathrm{m}\rangle, \langle \mathrm{a},\mathrm{c},\mathrm{l},\mathrm{m}\rangle, \langle \mathrm{d},\mathrm{e},\mathrm{l},\mathrm{m}\rangle$, $\langle \mathrm{u},\mathrm{u},\mathrm{l},\mathrm{m}\rangle, \langle \mathrm{v},\mathrm{r},\mathrm{l},\mathrm{m}\rangle$ □

We need to introduce predicates *IsNull* and *IsNotNull* in our formal treatment of nulls, because, as shown in Example 4.2, in the presence of `NULL`, SQL treats `IS NULL` and `IS NOT NULL` differently from classical $=$ and $\neq$, resp. For example, the queries

$$\mathcal{Q}(x)\colon \exists y(R(x,y) \wedge \mathit{IsNull}(y)) \ \text{ and } \ \mathcal{Q}'(x)\colon \exists y(R(x,y) \wedge y = \mathit{null}) \qquad (10)$$

are both conjunctive queries, but in SQL databases, they have different semantics.

4.2 Query answering under `NULL`*: the semantics*

Here we introduce the semantics of FO conjunctive query answering in relational databases with null values. More precisely, in SQL relational databases with a single null value, *null*, that is handled like the SQL `NULL`. We will exclude from the "SQL-like" conjunctive queries those such as (a) and (c) in Example 4.2. We will concentrate only on conjunctive queries with built-ins.

Definition 4.1
(a) $\mathit{Conj}(\Sigma^{\mathtt{null}})$ denotes the class of conjunctive queries in $\mathcal{L}(\Sigma^{\mathtt{null}})$ of the form

$$\mathcal{Q}(\bar{x}):\ \exists \bar{y}(A_1(\bar{x}_1) \wedge \cdots \wedge A_n(\bar{x}_n)), \qquad (11)$$

where $\bar{y} \subseteq \bigcup_i \bar{x}_i$, $\bar{x} = (\bigcup_i \bar{x}_i) \smallsetminus \bar{y}$, and the A_i are atoms containing any of the predicates in $\mathcal{R} \cup \mathcal{B}^{\mathtt{null}}$ plus terms, i.e. variables or constants in $\mathcal{U}$.
(b) $\mathit{Conj}^{\mathtt{sql}}(\Sigma^{\mathtt{null}})$ denotes the class of conjunctive queries as in (a) whose conjuncts are never of the form $t = \mathit{null}$, $t \neq \mathit{null}$, with t a term (*null* or not). □

[12] The same result is obtained from DBMSs that do not require an explicitly equality together with the join.

For example, for the queries in (10), $\mathcal{Q}, \mathcal{Q}' \in \mathit{Conj}(\Sigma^{\texttt{null}})$, $\mathcal{Q} \in \mathit{Conj}^{\texttt{sql}}(\Sigma^{\texttt{null}})$, but $\mathcal{Q}' \notin \mathit{Conj}^{\texttt{sql}}(\Sigma^{\texttt{null}})$. Notice that $\mathit{Conj}^{\texttt{sql}}(\Sigma^{\texttt{null}}), \mathit{Conj}(\Sigma) \subseteq \mathit{Conj}(\Sigma^{\texttt{null}})$. The idea is to force conjunctive queries *à la* SQL that explicitly mention the null value in (in)equalities, to use the built-ins InNull or $\mathit{IsNotNull}$.

Definition 4.2
Given a query in $\mathit{Conj}(\Sigma^{\texttt{null}})$ of the form $\mathcal{Q}(\bar{x})$: $\exists \bar{y}\psi(\bar{x}, \bar{y})$, with ψ quantifier-free, a variable v is *relevant* for $\mathcal{Q}$ if it occurs at least twice in ψ, without considering the atoms $\mathit{IsNull}(v)$, $\mathit{IsNotNull}(v)$, $v\theta \mathit{null}$, or $\mathit{null}\theta v$, with $\theta \in \mathcal{B}$. $\mathit{RelV}(\mathcal{Q})$ denotes the set of relevant variables for $\mathcal{Q}$. □

For example, for the query $\mathcal{Q}(x) : \exists y(P(x, y, z) \wedge Q(y) \wedge \mathit{IsNull}(y))$, $\mathit{RelV}(\mathcal{Q}(x)) = \{y\}$, because y is used twice in the subformula $P(x, y, z) \wedge Q(y)$.

As usual in FO logic, we consider assignments from the set, Var, of variables to the underlying database domain $\mathcal{U}$ (that contains constant null), i.e. $\sigma : Var \to \mathcal{U}$. Such an assignment can be extended to terms, mapping variables x to $\sigma(x)$, and $c \in \mathcal{U}$ to c. For an assignment σ, a variable y and a constant c, $\sigma\frac{y}{c}$ denotes the assignment that coincides with σ everywhere, except possibly on y, that takes the value c. Given a formula ψ, $\psi[\sigma]$ denotes the formula obtained from ψ by replacing its free variables by their values according to σ.

Now, given a formula (query) χ and an assignment σ, we verify if instance D satisfies $\chi[\sigma]$ by assuming that the quantifiers on relevant variables range over $(\mathcal{U} \smallsetminus \{\mathit{null}\})$, and those on non-relevant variables range over $\mathcal{U}$.

Definition 4.3
Let $\chi \in \mathit{Conj}(\Sigma^{\texttt{null}})$, and σ be an assignment. Instance D with σ *satisfies* χ *under the null-semantics*, denoted $D \models_N \chi[\sigma]$, exactly in the following cases: (below $t, t_1, \ldots$ are terms; and x, x_1, x_2 variables)

1. (a) $D \models_N \mathit{IsNull}(t)[\sigma]$, with $\sigma(t) = \mathit{null}$. (b) $D \models_N \mathit{IsNotNull}(t)[\sigma]$, with $\sigma(t) \neq \mathit{null}$.
2. $D \models_N (t_1 < t_2)[\sigma]$, with $\sigma(t_1) \neq \mathit{null} \neq \sigma(t_2)$, and $\sigma(t_1) < \sigma(t_2)$.[13]
3. (a) $D \models_N (x = c)[\sigma]$, with $\sigma(x) = c \in (\mathcal{U} \smallsetminus \{\mathit{null}\})$. (or symmetrically)[14]
   (b) $D \models_N (x_1 = x_2)[\sigma]$, with $\sigma(x_1) = \sigma(x_2) \neq \mathit{null}$.
   (c) $D \models_N (c = c)[\sigma]$, with $c \in (\mathcal{U} \smallsetminus \{\mathit{null}\})$.
4. (a) $D \models_N (x \neq c)[\sigma]$, with $\mathit{null} \neq \sigma(x) \neq c \in (\mathcal{U} \smallsetminus \{\mathit{null}\})$. (or symmetrically)
   (b) $D \models_N (c_1 \neq c_2)[\sigma]$, with $c_1 \neq c_2$, and $c_1, c_2 \in (\mathcal{U} \smallsetminus \{\mathit{null}\})$.
5. $D \models_N R(t_1, \ldots, t_n)[\sigma]$, with $R \in \mathcal{R}$, and $R(\sigma(t_1), \ldots, \sigma(t_n)) \in D$.
6. $D \models_N (\alpha \wedge \beta)[\sigma]$, with α, β quantifier-free, $\sigma(y) \neq \mathit{null}$ for every $y \in \mathit{RelV}(\alpha \wedge \beta)$, and $D \models_N \alpha[\sigma]$ and $D \models_N \beta[\sigma]$.
7. $D \models_N (\exists y\ \alpha)[\sigma]$ when: (a) if $y \in \mathit{RelV}(\alpha)$, there is c in $(\mathcal{U} \smallsetminus \{\mathit{null}\})$ with $D \models_N \alpha[\sigma\frac{y}{c}]$; or (b) if $y \notin \mathit{RelV}(\alpha)$, there is c in $\mathcal{U}$ with $D \models_N \alpha[\sigma\frac{y}{c}]$. □

This semantics also applies to $\mathit{Conj}^{\texttt{sql}}(\Sigma^{\texttt{null}})$.

[13] Of course, when there is an order relation on $\mathcal{U}$. We could introduce ">" similarly.
[14] Notice the use of the symbols $=$ and $\neq$ both at the object and the meta levels.

Definition 4.4
Let $\mathcal{Q}(\bar{x}) : \exists\bar{y}\psi(\bar{x}, \bar{y})$ be in $Conj(\Sigma^{\texttt{null}})$, with $\bar{x} = x_1, \ldots, x_n$, and ψ quantifier-free.
(a) A tuple $\langle c_1, \ldots, c_n\rangle \in \mathcal{U}^n$ is an *N-answer from D to* $\mathcal{Q}$, denoted $D \models_N \mathcal{Q}[c_1, \ldots, c_n]$, iff there is an assignment σ such that $\sigma(x_i) = c_i$, for $i = 1, \ldots, n$; and $D \models_N (\exists\bar{y}\psi)[\sigma]$.
(b) If $\mathcal{Q}$ is a sentence (a Boolean query), the N-answer is *yes* iff $D \models_N \mathcal{Q}$, and *no*, otherwise.
(c) $\mathcal{Q}^{Nsem}(D)$ denotes the set of N-answers to $\mathcal{Q}$ from instance D. □

Notice that $D \models_N (\exists\bar{y}\psi)[\sigma]$ in (a) above requires, according to Definition 4.3, that the relevant variables in the existential prefix $\exists\bar{y}$ do not take the value *null*. The free variables, i.e. in $\bar{x}$, may take the value *null* only when they are not relevant in the query. For illustration, in Example 4.1, since the free variable x is not relevant, $\mathcal{Q}_2^{Nsem}(D_2) = \{\langle null\rangle\}$.

Example 4.3
Consider the instance D and the conjunctive query below.

R	A	B
	a	b
	c	d
	e	$null$

S	B	C
	b	f
	d	g
	$null$	j

$$\mathcal{Q}(x, z)\colon\ \exists y(R(x, y) \wedge S(y, z)).$$

Here, under the classical semantics, $\mathcal{Q}(D) = \{\langle a, f\rangle, \langle c, g\rangle, \langle e, j\rangle\}$, treating *null* as any other constant. However, $\mathcal{Q}^{Nsem}(D) = \{\langle a, f\rangle, \langle c, g\rangle\} \subseteq \mathcal{Q}(D)$. □

It is easy to prove that, for queries in $Conj(\Sigma^{\texttt{null}})$: $\mathcal{Q}^{Nsem}(D) \subseteq \mathcal{Q}(D)$. Furthermore, the N-query answering semantics coincides with classical FO query answering semantics in databases without *null* . More precisely, if $null \notin \mathcal{U}$ (and then it does not appear in D or $\mathcal{Q}$ either): $D \models_N \mathcal{Q}[\bar{t}]$ iff $D \models \mathcal{Q}[\bar{t}]$.

4.3 Constraint satisfaction under NULL *via FO rewriting*

The notions of relevant attributes (or variables) and formula satisfaction under the null-semantics can be both extended to more complex formulas. In particular, they can be applied to constraint satisfaction under SQL NULL (Bravo and Bertossi 2006; Bravo 2007). As expected, the satisfaction of a constraint ψ by a database that may contain *null* depends upon the presence of *null* in the *relevant attributes* of ψ. The following is a generalization of Definition 4.2 to a larger class of formulas.

Definition 4.5
For $\psi \in \mathcal{L}(\Sigma^{\texttt{null}})$ in prenex normal form,[15] a variable v is *relevant* if it occurs at least twice in ψ, without considering occurrences in quantifiers or atoms of the forms $IsNull(v)$, $IsNotNull(v)$, $v\theta null$, or $null\theta v$, where θ a built-in comparison predicate. $RelV(\psi)$ denotes the set of relevant variables of ψ. □

[15] That is, of the form $\bar{Q}\chi$, where $\bar{Q}$ is a prefix of quantifiers, and χ is a quantifier-free formula. Actually, in this work all the formulas have a quantifier prefix of the form $\overline{\forall}\overline{\exists}$.

The constraints we are considering in this work, particularly those of the form (2), may not be in prenex normal form, but can be easily transformed while keeping the same variables and their occurrences, so that relevant variables can be determined.

Example 4.4
Consider the *referential integrity constraint* (RIC) on schema $\mathcal{R} = \{P(A, B, C), R(A, B, E)\}$: ψ: $\forall x \forall y \forall z (P(x, y, z) \rightarrow \exists v R(x, y, v))$; and the instance D:

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

R	A	B	E
	a	5	3
	a	3	7

DBMSs implement the so-called "simple semantics" of the SQL Standard for satisfaction of ICs. According to it, the database D above satisfies the RIC. This is because, for every tuple t in P, if $t[A]$ and $t[B]$ are different from *null*, there is a tuple t' in R with $t[A, B] = t'[A, B]$.

In this case, and informally, the attributes that are relevant for checking the satisfaction of the RIC (i.e. those we attempt to capture through the relevant variables) are A and B, in both P and R. If we try to insert tuple $P(c, d, null)$ into P, the DBMS will reject the insertion, because none of the attributes that are relevant for checking the constraint are *null*, and there is no tuple $R(c, d)$.

More precisely, the set of relevant variables for ψ is $RelV(\psi) = \{x, y\}$, because x and y appear twice in ψ. Accordingly, the values for attributes C and E are not relevant when checking the satisfaction of ψ, which makes sense. □

We will now formalize constraint satisfaction under the null-semantics. In this direction, we consider a single class of constraints that includes those in (1) and (2), and can be handled in a uniform manner. It also includes all the common constraints used in data management. More precisely, we consider constraints that are sentences in $\mathcal{L}(\Sigma^{\texttt{null}})$ of the form:

$$\forall \bar{x} (\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \longrightarrow \bigvee_{j=1}^{m} C_j), \qquad (12)$$

where $m, n \neq 0$, R_i is a predicate in $\mathcal{R}$, C_j is a conjunctive formula of the form $\exists \bar{y}_j \bigwedge_{k=1}^{l} Q_{jk}(\bar{x}'_{jk}, \bar{y}_{jk})$, where each Q_{jk} is a predicate in $\mathcal{R}$ or a built-in,[16] $\bar{x} = \bigcup_{i=1}^{n} \bar{x}_i$, $\bar{x}'_{jk} \subseteq \bar{x}$, and $\bar{y}_j = \bigcup_{k=1}^{l} \bar{y}_{jk}$.

Without loss of generality, we assume that the existentially quantified variables (the $\bar{y}_j$) do not appear in $\bar{x}$ and are different for each conjunctive literal C_j. Notice that (12) allows formulas with only built-in atoms in the consequent. We also assume that they do not have any *explicit* occurrence of the constant *null*.[17] Those formulas may contain the *IsNull* or *IsNotNull*, but as built-in predicates.

[16] Occurrences of variables in built-ins have to be *safe*, i.e. they also appear in $\bar{x}$ or in database predicate in the same conjunction.

[17] This is not an essential requirement, but will simplify the presentation. Furthermore, all reasonable DECs and local ICs do not require the explicit use of *null* as a constant. If we need to express a relational NOT-NULL-constraint, we can say, e.g. $\forall x \forall y (R(x, y) \rightarrow IsNotNull(x))$.

In order to define N-satisfaction of a constraint $\psi \in \mathcal{L}(\Sigma^{\texttt{null}})$ of the form (12), we first rewrite it into a new FO formula ψ^N, which makes explicit the role played by the relevant attributes in ψ (as in Definition 4.5) and the way nulls are handled in them. Next, satisfaction is defined in terms of the rewriting.

Definition 4.6
Let $\psi \in \mathcal{L}(\Sigma^{\texttt{null}})$ be a constraint of the form (12), i.e. $\forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \longrightarrow \bigvee_{j=1}^{m} C_j)$.
(a) *The N-rewriting of ψ* is the $\mathcal{L}(\Sigma^{\texttt{null}})$-sentence:

$$\psi^N:\ \ \forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \rightarrow (\bigvee_{v \in RelV(\psi)\cap \bar{x}} IsNull(v) \vee \bigvee_{j=1}^{m} C_j^N)), \tag{13}$$

with $\bar{x} = \cup_{i=1}^{n}\bar{x}_i$ and

$$C_j^N = \exists \bar{y}_j(\bigwedge_{k=1}^{l} Q_{jk}(\bar{x}'_{jk}, \bar{y}_{jk}) \ \ \wedge \bigwedge_{w \in ((RelV(\psi)\smallsetminus \bar{x})\cap \bar{y}_j)} IsNotNull(w))\cdot \tag{14}$$

(b) For an instance D, possibly containing *null*, ψ is *N-satisfied by D*, denoted $D \models_N \psi$, iff $D \models \psi^N$. Here, $D \models \psi^N$ refers to classical first-order satisfaction, with *null* treated as any other constant of the domain. □

We can see from Definition 4.6 that there are basically two cases when a ground instantiation of ψ (obtained by assigning constants to the variables in $\bar{x}$) is immediately satisfied due to the presence of *null*: (a) When *null* appears in any of the relevant attributes in the antecedent. (b) At least one of the conjunctive formulas C_j is true, considering that when they are checked according to equation (14), *null* is treated as any other constant, but the variables in existential joins do not take the value *null* (hence the condition based on $w \in (RelV(\psi) \smallsetminus \bar{x}) \cap \bar{y}_j$ in (14)). The rewriting can be applied in particular to our UDECs and RDECs, as follows.

For a UDEC ψ of the form (1), ψ^N is:

$$\forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \rightarrow (\bigvee_{v \in RelV(\psi)} IsNull(v) \vee \bigwedge_{k=1}^{l}(\bigvee_{j=1}^{m} Q_{kj}(\bar{y}_j))))\cdot \tag{15}$$

For an RDEC ψ of the form (2), ψ^N is:

$$\forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \ \rightarrow \ (\bigvee_{v \in (RelV(\psi)\cap \bar{x})} IsNull(v) \vee \exists \bar{y}(\bigwedge_{k=1}^{l}(Q_k(\bar{x}_k, \bar{y}_k) \wedge \varphi_k(\bar{x}'_k, \bar{y}'_k)) \wedge \bigwedge_{w \in (RelV(\psi)\smallsetminus \bar{x})} IsNotNull(w))\ \ )\ \ )\cdot \tag{16}$$

Example 4.5
(example 4.4 cont.) The rewriting ψ^N of the RIC ψ is, according to (16):

$$\psi^N:\ \forall x \forall y \forall z(P(x,y,z) \ \rightarrow \ IsNull(x) \vee IsNull(y) \vee \exists w R(x,y,w))\cdot$$

D classically satisfies ψ^N, treating *null* as any other constant. Then, $D \models_N \psi$. □

Example 4.6
For ψ: $\forall x(R(x) \rightarrow \exists y(T(x,y) \wedge S(y)),\ \ RelV(\psi) = \{x,y\}$. From (16):

$$\psi^N: \ \forall x(R(x) \ \ \rightarrow \ \ IsNull(x) \vee \exists y(T(x,y) \wedge S(y) \wedge IsNotNull(y))\cdot$$

(a) For $D = \{R(a)\}$, $D \not\models \psi^N$; and then, $D \not\models_N \psi$.
(b) For $D = \{R(a), T(a, null), S(null)\}$, $D \not\models \psi^N$; and then, $D \not\models_N \psi$.
(c) For $D = \{R(a), T(a,b), S(b)\}$, $D \models \psi^N$; and then, $D \models_N \psi$.
(d) For $D = \{R(null)\}$, $D \models \psi^N$; and then, $D \models_N \psi$.
(e) For $D = \emptyset$, $D \models \psi^N$; and then, $D \models_N \psi$. □

Example 4.7
Consider the schema $\mathcal{S} = \{R(A,B,E)\}$ with the primary *key constraint* (KC) $R: AB \rightarrow E$, expressing that attributes A, B, together, functionally determine attribute E.[18] An SQL database with the instance $D = \{R(a,5,3), R(a,3,7)\}$, which contains no nulls, would satisfy the KC.

The insertion of $R(null,4,5)$ into D would be rejected since the KC would be violated: *null* is not allowed in a key attribute.The insertion of $R(a,5,null)$ would also be rejected, but now due to the potential occurrence of two R-tuples t_1, t_2 with $t_1[A] = t_2[A] = a$ and $t_1[B] = t_2[B] = 5$, but being unknown whether $t_1[E] = t_2[E]$.

In order to fully capture this KC in predicate logic and the intended semantics we just described, we need the following sentences to represent the KC:

$$\psi_1: \ \forall x \forall y \forall z_1 \forall z_2 (R(x,y,z_1) \wedge R(x,y,z_2) \rightarrow z_1 = z_2), \quad (17)$$
$$\psi_2: \ \forall x \forall y \forall z_1 \forall z_2 (R(x,y,z_1) \wedge R(x,y,z_2) \wedge IsNull(z_1) \rightarrow IsNull(z_2)), \quad (18)$$
$$\psi_3: \ \forall x \forall y \forall z (R(x,y,z) \wedge IsNull(x) \rightarrow \mathbf{false}), \quad (19)$$
$$\psi_4: \ \forall x \forall y \forall z (R(x,y,z) \wedge IsNull(y) \rightarrow \mathbf{false})\cdot \quad (20)$$

The first two constraints ensure that, if two tuples coincide in attributes A and B, then they have the same value in z or they are both *null*. The last two constraints ensure that the values in attributes A or B in relation R cannot be *null*.

Actually, (18) can be written as

$$\forall x \forall y \forall z_1 \forall z_2 (R(x,y,z_1) \wedge R(x,y,z_2) \rightarrow \neg IsNull(z_1) \vee IsNull(z_2)),$$

or equivalently, as a UDEC of the form (1):

$$\psi_2': \ \forall x \forall y \forall z_1 \forall z_2 (R(x,y,z_1) \wedge R(x,y,z_2) \rightarrow IsNotNull(z_1) \vee IsNull(z_2)),$$

Similarly, (19) and (20) can be written as UDECs of the form (1):

$$\psi_3': \ \forall x \forall y \forall z (R(x,y,z) \ \rightarrow \ IsNotNull(x)),$$
$$\psi_4': \ \forall x \forall y \forall z (R(x,y,z) \ \rightarrow \ IsNotNull(y))\cdot$$

So, finally, the KC is represented by the UDECs $\psi_1, \psi_2', \psi_3', \psi_4'$.

[18] A key constraint is a particular kind of functional dependency, where a set of attributes of a relational predicate functionally determines all the attributes of the predicate. Declaring a key constraint as *primary* in an SQL-based relational DBMS has in particular the effect that `NULL` is not accepted in key attributes.

For illustration, considering that $RelV(\psi_1) = \{x, y, z_1, z_2\}$, and $RelV(\psi_3') = \emptyset$, we obtain from (15):

$$\begin{aligned}\psi_1^N:\ & \forall x \forall y \forall z_1 \forall z_2 (R(x,y,z_1) \wedge R(x,y,z_2) \ \rightarrow\ \mathit{IsNull}(x) \vee \mathit{IsNull}(y) \vee \\ & \qquad\qquad \mathit{IsNull}(z_1) \vee \mathit{IsNull}(z_2) \vee z_1 = z_2), \quad (21)\\ (\psi_3')^N:\ & \forall x \forall y \forall z (R(x,y,z) \ \rightarrow\ \mathit{IsNotNull}(x))\cdot \quad (22)\end{aligned}$$

Finally, observe that when the atom $z_1 = z_2$ in ψ_1^N is evaluated on the domain, the constant *null* is treated as any other constant, i.e. $\mathit{null} = c$ is true only when constant c is *null*. □

We have managed to reduce constraint satisfaction under `NULL` as handled by SQL databases to formula satisfaction in FO predicate logic, without assigning any special status to *null*, which is treated as an ordinary constant. We considered only constraints of the form (12), which include all our DECs and ICs for peers, in which case, the generic set of database predicates $\mathcal{R}$ will contain those for a single peer or a pair thereof. However, the semantics can be extended to more general FO formulas (Bravo 2007).

4.4 Query answering under `NULL` *via FO rewriting*

The notion of query answer under `NULL`, that of N-answer given in Section 4.2, being interesting *per se*, allowed us to motivate the treatment of constraints via relevant variables. However, query answering in presence of *null* can also be treated via the rewriting in Definition 4.6. This is achieved by considering conjunctive queries as a special case of (12), without the $\forall \bar{x}$ and empty antecedents; and obtaining their relevant variables through Definition 4.5.

More precisely, a conjunctive query $\mathcal{Q}(\bar{x}) \in Conj(\Sigma^{\mathtt{null}})$, i.e. of the form (11), is rewritten into a new conjunctive query as follows:

$$\mathcal{Q}^N(\bar{x}):\ \exists \bar{y}(A_1(\bar{x}_1) \wedge \cdots \wedge A_n(\bar{x}_n) \ \wedge \bigwedge_{v \in RelV(\mathcal{Q})} v \neq \mathit{null})\cdot \qquad (23)$$

It holds: $D \models_N \mathcal{Q}[\bar{c}]$ iff $D \models \mathcal{Q}^N[\bar{c}]$, where the latter is classic FO satisfaction, with *null* treated as an ordinary constant in the domain.

This transformation ensures that relevant variables range over $(\mathcal{U} \smallsetminus \{\mathit{null}\})$. $\mathcal{Q}^N(\bar{x}) \in Conj(\Sigma^{\mathtt{null}})$, and may contain atoms of the form $\mathit{IsNull}(t)$ or $\mathit{IsNotNull}(t)$. However, they can be replaced by $t = \mathit{null}$ or $t \neq \mathit{null}$, resp., leading to a query in $Conj(\Sigma)$, but with the same answers as (23).

Example 4.8
Consider the instance D below, and the query $\mathcal{Q}(x)\colon\ \exists y(P(x,y) \wedge y > 5)$, for which $RelV(\mathcal{Q}) = \{y\}$. In this case, the bound variable is the only relevant variable, and then ranges over non-null values when checking query satisfaction.

Accordingly, $D \models_N \mathcal{Q}[f]$ holds, because $\exists y(P(f, y) \wedge y > 5)$ is true in D, with 7 as a non-null value for y that makes the formula true. This result is confirmed by the rewriting of $\mathcal{Q}(x)$.

P	A	B
	f	7
	f	5
	null	8
	b	*null*

For $\mathcal{Q}^N(x)\colon\ \exists y(P(x, y) \wedge y > 5\ \wedge\ \mathit{IsNotNull}(y))$, $D \models \mathcal{Q}^N[f]$ holds, under classical satisfaction, with *null* treated as any other constant. Similarly, $D \models_N \mathcal{Q}[\mathit{null}]$. □

Example 4.9
(example 4.1 continued) The query in (9) can be rewritten as

$$\mathcal{Q}_2^N(x)\colon\ \exists y \exists z(R(x, y, z) \wedge S(y) \wedge y > 2 \wedge y \neq \mathit{null})\cdot$$

We had $D \not\models_N \mathcal{Q}_2[1]$. Now, $D \not\models \exists y \exists z(R(1, y, z) \wedge S(y) \wedge y > 2 \wedge y \neq \mathit{null})$, classically, with *null* treated as an ordinary constant. As expected, $D \not\models \mathcal{Q}_2^N[2]$ due to the new conjunct $y \neq \mathit{null}$.

Finally, $D \models \mathcal{Q}_2^N[\mathit{null}]$ because $D \models (R(\mathit{null}, 3, 3) \wedge S(3) \wedge 3 > 2 \wedge 3 \neq \mathit{null})$. Since *null* is treated as any other constant, we can compare it with 3. By the *unique names assumption*, $3 \neq \mathit{null}$ holds. □

Our query answering N-semantics for $\mathit{Conj}(\Sigma^{\mathtt{null}})$ can be applied in particular any SQL-like conjunctive query, but first expressing it as a query $\mathcal{Q}$ in $\mathit{Conj}^{\mathtt{sql}}(\Sigma^{\mathtt{null}})$, and then computing and classically evaluating $\mathcal{Q}^N$.

The notions of constraint satisfaction and query answer in the presence of `NULL` à la SQL we introduced in this section coincide with the classic notions in databases without *null*.

In the rest of this work, the notion of formula satisfaction that we have denoted with $\models_N$ will be simply denoted with $\models$. When constraints are not satisfied in this sense, we will apply a particular repair semantics that captures the special role of *null*. It is introduced in the next section.

5 Solution Semantics with NULL

In the preceding section, we introduced a notion of formula satisfaction with database instances that may contain *null*. We now use it for defining the semantics of a peer system where the data movement process relies on consistency restoration of peer instances with respect to DECs. The value *null* will be used to replace existentially quantified variables in consequents of referential DECs and local referential ICs as a possibility to consider for DEC and IC enforcement. Another possibility for the same task is tuple deletion.

The repair semantics that supports consistency restoration will have to capture and be sensitive to the (possibly multiple) presence of *null* in the database and its use for consistency enforcement. In particular, it has to give an account of the facts that:

1. When atoms are inserted into the database, an existential variable that appears in a join or in a built-in atom in an RDEC's consequent (a so-called *problematic existential variable*) is never replaced by (or takes the value) *null*.
2. Arbitrary non-null constants from the domain are not used for these problematic variables either. Actually, rather than introducing arbitrary values for existential variables of this kind, tuple deletions from antecedents in DECs will be privileged.

5.1 A restricted chase

In order to achieve the just stated goals, we first introduce a restricted, *ad hoc* form of the chase (Abiteboul et al. 1995), as an auxiliary construct. It is applied only with the DECs and ICs at hand that do not have problematic existential variables in their consequents. The enforcement of this subset of the constraints introduces tuples with *null* for existential variables (that do not appear in joins though).

In the end, this chase will return a finite set of atoms that will be used as a possibly generous upper-bound for the admissible tuple insertions that create repairs and solutions on the basis of the whole sets of DECs at hand.

As any other form of chase, our restricted chase enforces the satisfaction of constraints, but in our case the notion of satisfaction corresponds to that introduced in Section 4.3.

In the rest of this subsection, we consider a database instance D and a set of DECs $\Sigma = \Sigma_1 \cup \Sigma_2$, with Σ_1 a set of UDECs of the form (1) and Σ_2 a set of RDECs of the form (2). We consider the subset, Σ_2^- of Σ_2 that contains all the RDECs except for those involving existentially quantified variables in joins or built-in atoms. Accordingly, we consider $\Sigma^- := \Sigma_1 \cup \Sigma_2^-$.

Definition 5.1
The *restricted chase*, $r\text{-}Chase^{null}(D, \Sigma^-)$, with an instance D and $\Sigma^- = \Sigma_1 \cup \Sigma_2^-$, is the instance D' obtained as a fix-point of the following iterative procedure:

1. $D'_0 := D$.
2. Given instance D'_s, if $D'_s \models \Sigma^-$ (relative to $\models_N$), then $D'_{s+1} := D'_s$.
3. Given instance D'_s, if a ground instantiation, $\varphi\!\downarrow$, of a constraint $\varphi \in \Sigma_1$ (with constants or *null*) is not satisfied by D'_s (relative to $\models_N$), but its antecedent is, then D'_{s+1} is obtained from D'_s by adding the ground database atoms appearing in every disjunct of every conjunct in the consequent of $\varphi\!\downarrow$ (but only when the built-ins in the conjunct are satisfied).
   Notice that this excludes the generation of atoms with any kind of built-ins, including the atom **false**.
4. Given instance D'_s, if a ground instantiation (obtained by replacing universal variables with constants or *null*, but keeping the existential quantifiers) of a constraint in Σ_2^- is not satisfied by D'_s, but its antecedent is, then replace the existentially quantified variables by *null*, and build D'_{s+1} by adding to D'_s the corresponding ground atoms in the conjunct in the consequent (but only when the built-ins in it are satisfied). □

This procedure finitely terminates as we show below, but first some intuitions, explanations and examples. The chase procedure propagates and invents values due to the enforcement of RECs with non-problematic existential quantifiers and UDECs. As a result, the constants in the chase instance will be those already appearing in the initial instance D, those that appear explicitly in consequents of DECs, or *null* as a value invented for (non-problematic) existentially quantified variables.

Example 5.1
Consider the following set of constraints, classified according to Definition 5.1:

$$\begin{aligned}
\Sigma_1 \;=\; & \{\forall x \forall y (T(x,y) \rightarrow R(x,y)), & (24)\\
& \forall x \forall y \forall z (R(x,y) \wedge S(y,z) \rightarrow Q(x,y,z) \vee T(x,z)), & (25)\\
& \forall x \forall y \forall z (Q(x,y,z) \rightarrow S(x,y) \wedge R(y,z)), & (26)\\
& \forall x \forall y \forall z (T(x,y) \wedge T(x,z) \rightarrow y = z), & (27)\\
& \forall x \forall y (T(x,y) \wedge S(x,y) \rightarrow \mathbf{false}), & (28)\\
\Sigma_2^- \;=\; & \{\forall x \forall y (R(x,y) \rightarrow \exists z (Q(x,y,z) \wedge x \neq y), & (29)\\
& \forall x \forall y \forall z (Q(x,y,z) \rightarrow \exists w (R(x,z) \wedge S(x,w))\}\cdot & (30)
\end{aligned}$$

Independently from the instance at hand (initial or not) at a chase step, constraints (27) and (28), will never be applied (or enforced).

If D contains $T(a, null)$, then the ground instantiation $T(a, null) \rightarrow R(a, null)$ of (24) is satisfied under $\models_N$, even if D does not contain $R(a, null)$. This is because both variables x, y in (24) are relevant (cf. Definition 4.5). So, $R(a, null)$ is not generated. However, if $T(a,b) \in D$ and $R(a,b) \notin D$, the latter atom is generated.

Now, the ground instantiation $R(a,a) \rightarrow \exists z (Q(a,a,z) \wedge a \neq a)$ of (29) will not create new tuples because of the inequality $a \neq a$. However, if $R(a,b) \in D$, and the instantiation $R(a,b) \rightarrow \exists z (Q(a,b,z) \wedge a \neq b)$ is not satisfied, the tuple $Q(a,b,null)$ will be created.

The ground instantiation $R(a,b) \wedge S(b,c) \rightarrow\; Q(a,b,c) \vee T(a,c)$ of (25), with $R(a,b)$, $S(b,c) \in D$, will generate both $Q(a,b,c)$ and $T(a,c)$ (if it was not already satisfied).

Notice that creation of $Q(a,b,null)$ above, due to (29), feeds the antecedent of (26). However, its instantiation $Q(a,b,null) \rightarrow S(a,b) \wedge R(b,null))$ is satisfied, because *null* appear for variable z that is relevant. So, $R(b, null)$ is not generated. □

Remark 5.1
(a) Notice that in the DEC (26) all variables, x, y, z, are relevant. As a consequence, any ground atom with predicate Q with *null* in it will not trigger the rule. We can see that, every time the chase introduces *null* in a predicate position that turns out to be relevant in another DEC, this latter DEC will not be triggered (because it will be automatically satisfied). This possibly very common situation will contribute to a fast termination of the chase.

(b) We can assume that the chase applies in parallel all possible instantiations of

DECs in Σ^-. It is clear that this restricted *null*-based chase always terminates, because only the elements of the initial active domain plus possibly *null* are used to fill arguments in database predicates, which leads to a saturation point in polynomial-time in the size of instance D and the schema. In particular, the resulting instance is of polynomial size in the size of D.

(c) Instead of using the chase procedure (in case we wanted a more model-theoretic or declarative approach to the chase instance), we could replace each sentence in Σ_1 by a collection of Datalog rules, one for each disjunct in the consequent; and each sentence in Σ_2^- by a Datalog rule obtained replacing existential variables by *null*. The chase instance would coincide with the minimal model of the resulting Datalog program (Abiteboul et al. 1995). □

Notice that $r\text{-}Chase^{null}(D, \Sigma^-)$ may not satisfy Σ, for the trivial reason that not all DECs in Σ are considered in the chase. Actually, $r\text{-}Chase^{null}(D, \Sigma^-)$ may not even satisfy Σ^-, because it ignores DECs with only built-ins in consequents. For illustration, in Example 5.1, the chase of $D = \{T(a, b), T(a, c)\}$ will not satisfy (27), and the chase of $D = \{T(a, b), S(a, b)\}$ will not satisfy (28).

The next step will be imposing $r\text{-}Chase^{null}(D, \Sigma^-)$ as an upper bound on the extensions of possible repairs. With this we will discard candidate repairs that introduce arbitrary, non-mandatory, non-null constants. In other words, we will use the result of this chase to test candidate repairs (or solutions), demanding their inserted atoms to be contained in the chase applied with the subset Σ^- of the set Σ of DECs at hand.

5.2 Repair semantics with NULL and solutions

A repair semantics for single relational databases and sets of ICs that may include referential ICs is proposed in (Bravo and Bertossi 2006; Bravo 2007). It introduces null values of the kind described in Section 4.3. We will adopt and adapt this repair semantics in the context of PDESs, by taking into account the restrictions imposed by the chase we just introduced, and also the trust relationships.

So as repairs of an inconsistent instance with respect to a set of ICs, solutions for a peer in a PDES are expected to stay "close" to the original peer's instance, while satisfying its DECs. In order to capture "closeness" in our null-based semantics, we need to compare instances and their tuples, which may contain constant *null*.

Definition 5.2
(Levene and Loizou 1997) For constants $c, d \in \mathcal{U}$, c provides less or equal information than d, denoted $c \sqsubseteq d$, iff $c = null$ or $c = d$. For sequences $\bar{s}_1 = \langle c_1, \ldots, c_n\rangle$ and $\bar{s}_2 = \langle d_1, \ldots, d_n\rangle$, with $c_i, d_i \in \mathcal{U}$, $\bar{s}_1$ provides less or equal information than $\bar{s}_2$, denoted $\bar{s}_1 \sqsubseteq \bar{s}_2$, iff $c_i \sqsubseteq d_i$ for every $i = 1, \ldots, n$. Finally, $\bar{s}_1 \sqsubset \bar{s}_2$ means $\bar{s}_1 \sqsubseteq \bar{s}_2$ and $\bar{s}_1 \neq \bar{s}_2$. □

Definition 5.3
Consider D, D', D'' be database instances for the same schema, and a set Σ of DECs

of the form (1) or (2) in the language of the schema. Let Σ^- be the subset of Σ that excludes the DECs with problematic existential variables (cf. Section 5.1).

(a) *D' is at least as close to D as D''* (is to D), denoted $D' \leq_D^{\Sigma} D''$, iff one of the following holds:
   1. $D'' \not\subseteq$ *r-Chase*$^{null}(D, \Sigma^-)$.
   2. For every $P(\bar{a}) \in \Delta(D, D')$, there exists $P(\bar{a}') \in \Delta(D, D'')$, such that: i. $\bar{a} \sqsubseteq \bar{a}'$, and ii. if $\bar{a} \sqsubset \bar{a}'$, then $P(\bar{a}') \notin \Delta(D, D')$.

(b) *D' is closer to D than D''* (is to D), denoted $D' <_D^{\Sigma} D''$, iff $D' \leq_D^{\Sigma} D''$, but not $D'' \leq_D^{\Sigma} D'$. □

Definition 5.4
Given an instance D and a set Σ of DECs of the forms (1) or (2), a *null-based repair* of D with respect to Σ is an instance D' such that $D' \models_N \Sigma$, and there is no D'', such that $D'' \models_N \Sigma$ and $D'' <_D^{\Sigma} D'$. □

This definition ensures, in particular, that a database that, due to the enforcement of an RDEC with a non-problematic existential variable, inserts a tuple with *null* for that variable, is closer to D than another that adds some other, arbitrary constant. At the same time, the definition makes us prefer repairs obtained via tuple deletions instead of tuple insertions (with values null or not), when enforcing RDECs with problematic existential variables.

The null-based repair semantics, in particular Definition 5.4, is also of independent interest in the context of single inconsistent databases with respect to a set of ICs that are like the DECs we are considering in the PDES setting. Actually, this is the official and detailed formalization of the repair semantics first sketched and used in (Bravo and Bertossi 2006) for databases with referential ICs. The idea in that case is that inconsistencies are preferably repaired through the insertion of tuples with the special constant *null*, that is used as the null in SQL. When this is not possible, tuple deletions are preferred. This happens either because a constraint cannot be solved via tuple insertions at all (e.g. functional dependencies) or an inclusion dependency would have to be satisfied through the use of *null* for variables in a join. The next two examples show how this repair semantics would be used for repairing a single database.

Example 5.2
(example 4.6 cont.) Consider $\Sigma = \{\forall x(R(x) \rightarrow \exists y(T(x,y) \wedge S(y))\}$, and the instance $D = \{R(a)\}$.

Here, $\Sigma^- = \emptyset$, and *r-Chase*$^{null}(D, \Sigma^-) = \{R(a)\}$. Furthermore, $D \not\models \Sigma$ (this is case (a) in Example 4.6).

i. $D_1' = \{R(a), T(a, null), S(null)\} \not\models \Sigma$ (this is case (b) in Example 4.6). So, D_1' is not a repair.

ii. $D_2' = \emptyset \models \Sigma$ (this is case (e) in Example 4.6). D_2' is a repair.
Indeed, for any instance D'' that is not contained in *r-Chase*$^{null}(D, \Sigma^-)$, $D_2' \leq_D^{\Sigma} D''$ holds. So, the only possible improvement on D_2' could be $\{R(a)\}$, which does not satisfy Σ.

iii. $D'_3 = \{R(a), T(a,a), S(a)\} \models \Sigma$ (as case (c) in Example 4.6), but it is not a repair.
Indeed, first notice that $D'_2 \leq^{\Sigma}_{D} D'_3$, because $D'_3 \not\subseteq$ *r-Chase*$^{null}(D, \Sigma^-)$.
Now, $D'_3 \not\leq^{\Sigma}_{D} D'_2$, because condition 2. of Definition 5.3 does not hold for $S(a)$ (or $T(a,a)$). We have obtained that $D'_2 <^{\Sigma}_{D} D'_3$, i.e. the former, which is a repair, is strictly closer to D than the latter. The same holds for any instance of the form $D'_4 = \{R(a), T(a,b), S(b)\}$ (and its supersets). □

Example 5.3
Consider $\Sigma = \{\forall x \forall y \forall z (T(x,y) \wedge T(x,z) \rightarrow y = z),\ \forall x \forall y (T(x,y) \wedge S(x,y) \rightarrow$ **false**$)\}$, containing a functional dependency and a denial constraint. The instance $D = \{T(a,b), T(a,c), S(a,c)\}$ is inconsistent with respect to Σ.

Here, $\Sigma^- = \Sigma$. However, the restricted chase does not apply any of the ICs, because they contain only built-ins in the consequent. So, *r-Chase*$^{null}(D, \Sigma^-) = D$. As a consequence, only repairs based on tuple deletions are acceptable. In this case, $D' = \{T(a,b), S(a,c)\}$ is the only repair. □

Since the $\leq^{\Sigma}_{D}$ relations can take the place of the generic relation $\preceq^{\Sigma(\texttt{P})}_{D}$ used in Section 3, $\leq_D$ determines, through the instantiation of Definition 3.1, a concrete repair semantics that determines null-based neighborhood solutions for a peer `P`, in which case the definition has to be applied with $\Sigma(\texttt{P})$. However, Definition 5.4 does not take into account the trust relationships in a neighborhood that are imposed in Definition 3.1. As a consequence, to have a null-based notion of neighborhood solution, we have to impose the additional requirement that certain database relations in a neighborhood instance have to stay untouched (they belong to a peer that is most trusted).

Having a specific notion of neighborhood solution, it is possible to obtain, following the developments in Section 3, also specific definitions of solution and core for a peer, and the notion of peer-consistent answer. In the rest of this section we provide some additional examples that show aspects of this specific semantics, and in the following subsections we investigate computational problems related to this semantics.

Example 5.4
(example 5.2 continued) Consider a PDES instance $\mathfrak{D}$ for the schema $\mathfrak{P}$ with peers `P1` and `P2`, with $\mathcal{S}(\texttt{P1}) = \{R\}$, $\mathcal{S}(\texttt{P2}) = \{T, S\}$, and *Trust* $= \{(\texttt{P1}, \mathit{same}, \texttt{P2})\}$. Assume the only DECs are those in $\Sigma(\texttt{P1},\texttt{P2}) = \{\forall x (R(x) \rightarrow \exists y (T(x,y) \wedge S(y))\}$. So, $\Sigma^-(\texttt{P1},\texttt{P2}) = \emptyset$.

Assume that the instance of the neighborhood of `P1` is $D = \{R(a)\}$. Here, $D \not\models \Sigma(\texttt{P1},\texttt{P2})$. So, in order to obtain a neighborhood solution, we have to repair D with respect to $\Sigma(\texttt{P1},\texttt{P2})$.

Since `P1` trusts `P2` the same as itself, in principle we could insert tuples into `P2`'s relations or delete tuples from `P1`'s relation. However, in this case (cf. Example 5.2), since *r-Chase*$^{null}(D, \Sigma^-(\texttt{P1},\texttt{P2})) = \{R(a)\}$, the only neighborhood solution is the empty instance: $\emptyset \models_N \Sigma(\texttt{P1},\texttt{P2})$, and $\emptyset <^{\Sigma(\texttt{P1})}_{D} D'$, for any $\mathcal{S}(\texttt{P1},\texttt{P2})$-instance D' that inserts tuples. □

Similarly, the generic Definition 3.2 of solution and core for peers, can be instantiated with the null-based solution semantics, i.e. on the basis of the $\leq^{\Sigma}_{D}$ relations, obtaining specific versions of those definitions. Accordingly, we recall that, given an instance $\mathfrak{D} = \{D(\mathtt{Q}) \mid \mathtt{Q} \in \mathcal{P}\}$ for the schema $\mathfrak{P}$, the *core* of peer P is the intersection of its solutions: $Core(\mathtt{P}, \mathfrak{D}) := \bigcap Sol(\mathtt{P}, \mathfrak{D})$.

Example 5.5
(example 2.1 continued) Consider the following peers' instances: $D(\mathtt{P1})$=$\{R^1(a, 2)\}$, $D(\mathtt{P2}) = \{R^2(c, 4), R^2(d, 5)\}$, $D(\mathtt{P3}) = \{R^3(c, 4)\}$, and $D(\mathtt{P4}) = \{R^4(d, 5, 1)\}$. So, the PDES instance is $\mathfrak{D} = \{D(\mathtt{P1}), D(\mathtt{P2}), D(\mathtt{P3}), D(\mathtt{P4})\}$.

If we want the solutions for P4, we first need the solutions for P3 and P2, who also needs the solution for P3. Since P3 has no DECs with other peers, its only neighborhood solution is its local instance $D(\mathtt{P3})$, which is sent back to P2 or P4 if request.

Peer P2 needs to find the neighborhood solutions for $\{R^2(c, 4), R^2(d, 5), R^3(c, 4)\}$ with respect to $\Sigma(\mathtt{P2}, \mathtt{P3})$. Since P2 trusts P3 the same as itself, it can modify its own data or the data it got from P3. There are two neighborhood solutions for P2: $\{R^2(c, 4), R^2(d, 5)\}$ and $\{R^2(d, 5), R^3(c, 4)\}$, that restricted to P2's schema lead to two solutions for P2: $Sol(\mathtt{P2}, \mathfrak{D})$=$\{\{R^2(c, 4), R^2(d, 5)\}, \{R^2(d, 5)\}\}$. Peer P2 sends to P4 the intersection of its solutions: $Core(\mathtt{P2}, \mathfrak{D}) = \{R^2(d, 5)\}$.

Neighborhood solutions for P4 are obtained by repairing $\{R^4(d, 5, 1), R^2(d, 5), R^3(c, 4)\}$ with respect to $\Sigma(\mathtt{P4}, \mathtt{P2})$, and $\Sigma(\mathtt{P4}, \mathtt{P3})$. The DECs in $\Sigma(\mathtt{P4}, \mathtt{P2})$ are already satisfied, but not those in $\Sigma(\mathtt{P4}, \mathtt{P3})$. Since P4 trusts the data in P3 more than its own, the only neighborhood solution for P4 is obtained by inserting a tuple with *null* into P4's: $\{R^4(d, 5, 1), R^2(d, 5), R^3(c, 4), R^4(c, 4, null)\}$. Consequently, $Sol(\mathtt{P4}, \mathfrak{D}) = \{\{R^4(d, 5, 1), R^4(c, 4, null)\}\}$. $\Box$

Due to a particular combination of DECs, trust relationships, and local instances, it is possible that a peer has not solution, which may happen with or without the null-based repair semantics. For illustration, Example 3.2, now with the null-based solution semantics, still does not have solutions.

5.3 Complexity of neighborhood solutions

We now investigate the complexity of decision problems related to the general case of DECs of the form (12) (cf. Section 4), in combination with the null-based repair semantics we introduced in this section.

We concentrate mostly on the case that is directly relevant to our forthcoming answer-set programming (ASP) approach to specifying solutions for individual peers when they have already gathered their neighbors solutions. That is, we consider specifications and reasoning at the neighborhood level (cf. Section 6). Accordingly, we start by analyzing the complexity of deciding if an instance is a neighborhood solution (cf. Definition 3.1).

Definition 5.5
Consider a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, Trust \rangle$ and a peer $\mathtt{P} \in \mathcal{P}$. Given an

instance $\bar{D}$ for the neighborhood schema $\mathcal{S}(\mathcal{N}(\mathtt{P}))$ around $\mathtt{P}$, and an instance J for the schema $\mathcal{S}(\mathcal{N}(\mathtt{P}))$, the $\mathsf{NeighborhoodSol}$ decision problem is about determining if J is a neighborhood solution for $\mathtt{P}$ and $\bar{D}$, i.e. about membership of the set:

$\mathsf{NeighborhoodSol}(\mathfrak{P}, \mathtt{P}) = \{(J, \bar{D}) \mid J \in \mathit{NS}(\mathtt{P}, \bar{D})\}$. $\square$

This decision problem is parameterized by peer schemas and selected peers. The inputs are database instances. Then, we are considering data complexity.

Proposition 5.1
$\mathsf{NeighborhoodSol}$ is *coNP*-complete in the size of $J \cup \bar{D}$. $\square$

Notice that Proposition 5.1 has to be interpreted as follows (and similarly those that follow in this section): For every peer schema $\mathfrak{P}$, $\mathsf{NeighborhoodSol}(\mathfrak{P}, \mathtt{P})$ is in *coNP*; and there is a peer schema $\mathfrak{P}_0$ and a peer $\mathtt{P}_0$ in it, such that the associated problem $\mathsf{NeighborhoodSol}(\mathfrak{P}_0, \mathtt{P}_0)$ is *coNP*-hard.

Proposition 5.1 holds already for PDES with a single peer with local integrity constraints; actually with (cyclic) sets of RDECs of the simple form $\forall \bar{x}(R(\bar{x}) \longrightarrow \exists \bar{y}\, Q(\bar{x}', \bar{y}))$. The proof in this case is a bit more involved since tuple deletions and insertions are in principle possible, but the latter were blocked with the trust relationships. A proof can be found in (Bravo 2007).[19] The proposition also holds in the case where $\bar{D} = \bigcup_{\mathtt{Q} \in \mathcal{N}(\mathtt{P})} J_{\mathtt{Q}}$, but every $J_{\mathtt{Q}}$ with $\mathtt{Q} \in \mathcal{N}^{\circ}(\mathtt{P})$ is fixed, i.e. only J and $J_{\mathtt{P}}$ vary.

5.4 Complexity of the core and peer-consistent answers

Definition 5.6
Consider a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust} \rangle$ and a peer $\mathtt{P}$ in it. Given a neighborhood instance $\bar{D} = \bigcup_{\mathtt{Q} \in \mathcal{N}(\mathtt{P})} J_{\mathtt{Q}}$ for $\mathcal{S}(\mathcal{N}(\mathtt{P}))$, the *local core* of $\mathtt{P}$ is the intersection of the neighborhood solutions for $\mathtt{P}$ and $\bar{D}$, but restricted to $\mathtt{P}$'s schema $\mathcal{S}(\mathtt{P})$:
$\mathit{localCore}(\mathtt{P}, \bar{D}) := (\bigcap \mathit{NS}(\mathtt{P}, \bar{D}))\downharpoonright \mathcal{S}(\mathtt{P})$. $\square$

In this definition, $J_{\mathtt{Q}}$ denotes an arbitrary instance for peer $\mathtt{Q}$, which may be different from what we have called the initial instance $D(\mathtt{Q})$ for $\mathtt{Q}$ (cf. Definition 3.2). Actually, $J_{\mathtt{Q}}$ could be $D(\mathtt{Q})$, but also a neighborhood solution for $\mathtt{Q}$ restricted to its schema, or the intersection of the latter, etc. In this regard, we recall that a neighborhood solution for $\mathtt{P}$ is defined in terms of its neighborhood schema, its local instance $D(\mathtt{P})$ in $\mathfrak{D}$, and instances $J_{\mathtt{Q}}$ for other peers $\mathtt{Q}$ in its neighborhood (cf. Definition 3.1). Each $J_{\mathtt{Q}}$ for a neighboring peer $\mathtt{Q}$ is typically the restriction to $\mathcal{S}(\mathtt{Q})$ of the intersection of $\mathtt{Q}$'s local neighborhood solutions, which may not necessarily be the same as the initial instance $D(\mathtt{Q}) \in \mathfrak{D}$.

We can see that the problems of defining and computing neighborhood solutions for a peer can be formulated with arbitrary instances for the neighbors, and do not require taking into consideration the recursive relations between peers. However,

[19] It is also possible to modify the proof just given by introducing auxiliary, dummy joins in the consequent of the RDEC in $\Sigma(\mathtt{P1}, \mathtt{P1})$, to make attributes relevant and avoid so the insertion of tuples with *null*, which has the effect of enforcing deletions.

the computation of the solutions of a peer is a global problem that does include recursion. In this case, we fix all the instances to be the initial ones for each peer.

Notice from Definition 3.2, that when $\bar{D} = D(\mathtt{P}) \cup \bigcup_{\mathtt{Q} \in \mathcal{N}^\circ(\mathtt{P})} Core(\mathtt{Q}, \mathfrak{D})$ in Definition 5.6, it holds $localCore(\mathtt{P}, \bar{D}) = Core(\mathtt{P}, \mathfrak{D})$. In particular, Proposition 5.1 remains true if $J_{\mathtt{Q}} = Core(\mathtt{Q}, \mathfrak{D})$, for $\mathtt{Q} \in \mathcal{N}^\circ(\mathtt{P})$, and $J_{\mathtt{P}} = D(\mathtt{P})$.

Definition 5.7
Consider a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, Trust \rangle$ and a peer $\mathtt{P} \in \mathcal{P}$, the $\mathsf{InLocalCore}$ decision problem is about membership of the set:

$$\mathsf{InLocalCore}(\mathfrak{P}, \mathtt{P}) = \{(R(\bar{t}), \bar{D}) \mid R(\bar{t}) \in localCore(\mathtt{P}, \bar{D}) \text{ and } \bar{D} \text{ is an instance for } \mathcal{S}(\mathcal{N}(\mathtt{P}))\}. \qquad \square$$

Proposition 5.2
For a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, Trust \rangle$ and a peer $\mathtt{P} \in \mathcal{P}$, $\mathsf{InLocalCore}(\mathfrak{P}, \mathtt{P})$ is Π^P_2-complete in the size of $\bar{D}$, the input neighborhood instance. $\square$

Despite the assumption that the peer graphs $\mathcal{G}(\mathfrak{P})$ are acyclic (cf. Remark 2.1), sets Σ of DECs may contain cycles through inclusion dependencies, in particular with existential quantifiers, i.e. RDECs.

More precisely, assume Σ is (or contains) a set of sentences φ of the form $\forall \bar{x}(\psi(\bar{x}) \rightarrow \exists \bar{y} \chi(\bar{x}, \bar{y}))$, of the form (12). When $\bar{y}$ is not empty, φ is called *existential*. A directed graph can be associated to Σ. The nodes are the database predicates, and there is an edge from predicate P to predicate Q when P appears in the antecedent, and Q in the consequent of a $\varphi \in \Sigma$. The edge is *marked* if φ is existential. Σ is *ref-acyclic* if there no cycles with marked edges in the graph.[20]

For example, $IC_1 = \{\forall x(S(x) \rightarrow Q(x)), \forall x(Q(x) \rightarrow \exists y\ T(x, y))\}$ and $IC_2 = \{\forall x(S(x) \rightarrow Q(x)), \forall x(Q(x) \rightarrow S(x))\}$ are ref-acyclic sets, whereas $IC_3 = IC_1 \cup \{\forall xy\ (T(x, y) \rightarrow Q(y))\}$ is not. This notion can be applied without changes to the sets $\Sigma(\mathtt{P})$ in PDESs.

Example 5.6
Given a PDES $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, Trust \rangle$ with $\mathcal{P} = \{\mathtt{P1}, \mathtt{P2}\}$, $\mathfrak{S} = \{\mathcal{S}(\mathtt{P1}), \mathcal{S}(\mathtt{P1})\}$, $\mathcal{S}(\mathtt{P1}) = \{R^1(\cdot, \cdot)\}$, $\mathcal{S}(\mathtt{P2}) = \{R^2(\cdot, \cdot)\}$, $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall x \forall z (R^1(x, z) \rightarrow \exists y R^2(x, y))$, $\forall x \forall z (R^2(x, z) \rightarrow \exists y R^1(x, y)\}$, $\Sigma(\mathtt{P2}, \mathtt{P1}) = \emptyset$; and $Trust = \{(\mathtt{P1}, less, \mathtt{P2})\}$. In this case, the peer graph $\mathcal{G}(\mathfrak{P})$ is acyclic. However, schema $\mathfrak{P}$ is not ref-acyclic, because $\Sigma(\mathtt{P1}) = \Sigma(\mathtt{P1}, \mathtt{P2})$ has a cycle through RDECs. $\square$

In some cases we will make the assumption that the sets of DECs at hand are *ref-acyclic*. For this reason, we make notice that the proof of Proposition 5.2 uses a ref-acyclic set of DECs. So, the proposition still holds for this class of DECs, which will become relevant in Section 6. (Cf. the Online Appendix A for an additional discussion.)

[20] The condition of ref-acyclicity was already used in (Bravo and Bertossi 2006), as *RIC-acyclicity*, for sets of ICs on a single database schema.

Corollary 5.1
For a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust} \rangle$ and a peer $\mathtt{P} \in \mathcal{P}$, with ref-acyclic sets of DECs, the $\mathsf{InLocalCore}(\mathfrak{P}, \mathtt{P})$ decision problem is Π^P_2-complete in the size of $\bar{D}$, the input neighborhood instance. □

We can also apply the null-based solution semantics to Definition 3.3, obtaining a specific definition of *peer consistent answer*. More precisely, given a PDES schema $\mathfrak{P}$, an instance $\mathfrak{D}$ for it, a peer $\mathtt{P}$ and a query $\mathcal{Q}(\bar{x}) \in \mathcal{L}(\mathtt{P})$, the set of *peer consistent answers* to $\mathcal{Q}$ from $\mathtt{P}$ is

$$PCA^{\mathfrak{D}}_{\mathfrak{P},\mathtt{P}}(\mathcal{Q}) = \{t \mid D \models_N \mathcal{Q}[t], \text{ for all } D \in \mathit{Sol}(\mathtt{P}, \mathfrak{D}) \ \}, \qquad (31)$$

with $\mathit{Sol}(\mathtt{P}, \mathfrak{D})$ as defined in this section, on the basis of the $\leq^{\Sigma(\mathtt{P})}_D$ relations.

Proposition 5.2 and its Corollary 5.1 still hold when, for each $\mathtt{Q} \in \mathcal{N}^\circ(\mathtt{P})$, $J_{\mathtt{Q}} = \mathit{Core}(\mathtt{Q}, \mathfrak{D})$, and $J_{\mathtt{P}} = D(\mathtt{P})$. Therefore, the local computation of peer consistent answers to a conjunctive query -for which the cores of the neighboring peers are used- is also Π^P_2-complete.

Corollary 5.2
For a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust} \rangle$, an instance $\mathfrak{D}$ for $\mathfrak{P}$, and a peer $\mathtt{P} \in \mathcal{P}$, deciding answers to a conjunctive query posed to $\mathtt{P}$ that are true in $\bigcap NS(\mathtt{P}, \bar{D})$, where $\bar{D} = D(\mathtt{P}) \cup \bigcup_{\mathtt{Q} \in \mathcal{N}^\circ(\mathtt{P})} \mathit{Core}(\mathtt{Q}, \mathfrak{D})$, is Π^P_2-complete in the size of $\bar{D}$. This is also true for $\mathfrak{P}$ with ref-acyclic sets of DECs. □

The complexity of the decision and computational problems at the neighborhood level we have obtained so far are the most interesting, due to the modular and local manner solutions and peer-consistent answers are computed. These results will be useful in Section 6, where specification and computation of neighborhood solutions and PCAs are addressed.

We make only some final remarks in relation to global solutions and PCAs on their basis. Corollary 5.2 already tells us that deciding peer-consistent answers (which involves global solutions as opposed to neighborhood solutions) will be at least Π^P_2-hard. Actually, using Proposition 5.2 it is possible to provide a non-deterministic polynomial time algorithm (in data) with a Π^P_2-oracle to decide if an instance for a peer is a (global) solution for the peer. On this basis, one can obtain that PCA, as a decision problem, belongs to Π^P_3, in data (that includes those of all peers in the system).

5.5 The import case

In this section we consider a common situation, namely *the import case*, where we find *Trust* relationships are only of the form $(\mathtt{P}, \mathit{less}, \mathtt{Q})$ when $\mathtt{P} \neq \mathtt{Q}$; and the DECs are used for importing data from other peers.

Definition 5.8
Consider a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust} \rangle$, and two different peers $\mathtt{P}$ and $\mathtt{Q}$:

(a) An *import UDEC* (IUDEC) *to* P *from* Q is a UDEC in $\mathcal{L}(\mathtt{P},\mathtt{Q})$ of the form:

$$\forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \longrightarrow (Q(\bar{x}') \vee \varphi(\bar{x}''))), \tag{32}$$

where the R_i are predicates in $\mathcal{S}(\mathtt{Q})$, Q is a predicate in $\mathcal{S}(\mathtt{P})$, $\bar{x}', \bar{x}'' \subseteq \cup \bar{x}_i = \bar{x}$, and $\varphi(\bar{x}'')$ is a disjunction of built-ins (representing a conjunction of built-ins on the variables in the antecedent).

(b) An *import RDEC* (IRDEC) *to* P *from* Q is an RDEC in $\mathcal{L}(\mathtt{P},\mathtt{Q})$ of the form:

$$\forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \longrightarrow \exists \bar{z}(Q(\bar{y}, \bar{z}) \wedge \varphi_1(\bar{x}', \bar{z}')) \vee \varphi_2(\bar{x}'')), \tag{33}$$

where the R_i are predicates in $\mathcal{S}(\mathtt{Q})$, Q is a predicate in $\mathcal{S}(\mathtt{P})$, $\bar{x}', \bar{x}'', \bar{y} \subseteq \cup_i \bar{x}_i = \bar{x}$, $\bar{z}' \subseteq \bar{z}$, and φ_1 (φ_2) is a conjunction (disjunction) of built-ins. Notice that φ_1 basically imposes conditions with built-ins on the values for $\bar{z}$ that, under the *null*-based semantics, will be all *null*. ($\varphi_2(\bar{x}'')$ plays the same role on φ in (32).)

(c) A PDES is of the *import kind* if, for every peer P, the DECs in $\Sigma(\mathtt{P},\mathtt{Q})$ with any peer Q different from P, are import DECs to P from Q. Furthermore, if this set of DECs is non-empty, $(\mathtt{P}, \mathit{less}, \mathtt{Q}) \in \mathit{Trust}$. □

Notice that this definition does not make any assumptions on the possible sets of local constraints, $\Sigma(\mathtt{P},\mathtt{P})$.

Example 5.7
Consider a PDES with $\mathcal{P} = \{\mathtt{P1},\mathtt{P2},\mathtt{P3},\mathtt{P4}\}$, $\mathcal{S}(\mathtt{P1}) = \{R^1(\cdot,\cdot)\}, \mathcal{S}(\mathtt{P2}) = \{R^2(\cdot,\cdot), S^2(\cdot,\cdot)\}$, $\mathcal{S}(\mathtt{P3}) = \{R^3(\cdot,\cdot)\}, \mathcal{S}(\mathtt{P4}) = \{R^4(\cdot,\cdot,\cdot)\}$, and the following sets of DECs:

$$\begin{array}{rcl}
\Sigma(\mathtt{P1},\mathtt{P2}) &=& \{\forall x \forall y (R^2(x,y) \rightarrow (R^1(x,y) \vee x \leq y))\}, \\
\Sigma(\mathtt{P4},\mathtt{P2}) &=& \{\forall x \forall y \forall z (R^2(x,y) \wedge S^2(y,z) \ \rightarrow \ R^4(x,y,z))\}, \\
\Sigma(\mathtt{P4},\mathtt{P3}) &=& \{\forall x \forall y (R^3(x,y) \rightarrow \exists z R^4(x,y,z))\} \cdot \\
\Sigma(\mathtt{P3},\mathtt{P2}) &=& \{\forall x \forall y (R^3(x,y) \rightarrow R^2(x,y))\} \cdot
\end{array}$$

The first three sets of DECs are formed by import DECs, but not the last one. □

5.5.1 The unrestricted case

The *unrestricted import case* of PDES $\mathfrak{P}$ as in Definition 5.8 occurs when, for every peer $\mathtt{P} \in \mathcal{P}$, $\Sigma(\mathtt{P},\mathtt{P}) = \emptyset$. That is, in this case, a peer may have DECs of the forms (32) and (33) to neighboring peers, in whom it has more trust than in itself, but no local ICs.

As a consequence of an unrestricted local repair process, a peer will get data from its neighbors and will integrate them, at least virtually, into its own neighborhood instance. The data from a neighboring peer will be obtained by just posing it a conjunctive query, the one corresponding to the antecedent of the DEC.

For illustration, in Example 5.7, if P4 uses its import DEC from P2 (that in $\Sigma(\mathtt{P4},\mathtt{P2})$) to retrieve data from P2, then P4 sends to P2 the query: $\mathcal{Q}(x,y) : R^2(x,y) \wedge S^2(y,z)$.

In the import case we can define a Datalog program for which its minimal model is the neighborhood solution of the peer.

Definition 5.9
Given an unrestricted import PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$, a peer $\mathtt{P} \in \mathcal{P}$ and an instance $\bar{D}$ for the neighborhood schema $\mathcal{S}(\mathcal{N}(\mathtt{P}))$ around $\mathtt{P}$, *the import program* $\mathfrak{I}(\mathtt{P}, \bar{D})$ is a Datalog program containing:

1. The facts: $R(\bar{a})$, for every atom $R(\bar{a}) \in \bar{D}$.
2. For every IUDEC in $\Sigma(\mathtt{P})$ of the form (32), the rule:
$$Q(\bar{y}) \leftarrow R_1(\bar{x}_1), \ldots R_n(\bar{x}_n), \neg\varphi(\bar{x}'').$$
3. For every import RDEC in $\Sigma(\mathtt{P})$ of the form (33), the rule:
$$Q(\bar{y}, \overline{null}) \leftarrow R_1(\bar{x}_1), \ldots R_n(\bar{x}_n),\ \neg\varphi_2(\bar{x}'').$$

Here, $\neg\varphi(\bar{x}'')$ and $\neg\varphi_2(\bar{x}'')$ become conjunctions of built-in literals, i.e. atomic formulas with a built-in or negations thereof; and $\overline{null}$ is a sequence of nulls for variables $\bar{z}$. According to the discussion of (33), conditions associated to φ_1 become conditions on $\bar{x}'$, which can all be made part of $\bar{x}''$. □

Proposition 5.3
Given an unrestricted import PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$ and an instance $\mathfrak{D}$ over it, every peer $\mathtt{P} \in \mathcal{P}$ has a unique solution instance. Furthermore, there is an algorithm, that uses the import program $\mathfrak{I}(\mathtt{P}, D')$, and computes the solution for a peer $\mathtt{P}$ in polynomial time in the size of $\mathfrak{D}{\downharpoonright}\mathcal{S}(\mathcal{AC}(\mathtt{P}))$. □

This uniqueness result relies on the fact that we are repairing IRDECs through the insertion of null values, as sanctioned by the official repair semantics. Otherwise, uniqueness would hold in general only for IUDECs.

5.5.2 The restricted case

This case appears when, under all the import assumptions above, peers are allowed to have local constraints, that is, it may be that $\Sigma(\mathtt{P}, \mathtt{P}) \neq \emptyset$. In this case, a peer will import data without restrictions from its neighbors, but when building neighborhood solutions, also the local ICs will be taken into account. In this case, P may have none or several solutions.

Example 5.8
Consider the PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$ corresponding to Figure 7. Here, $\mathcal{P} = \{\mathtt{P1}, \mathtt{P2}, \mathtt{P3}\}$, $\mathfrak{S} = \{\mathcal{S}(\mathtt{P1}), \mathcal{S}(\mathtt{P2}), \mathcal{S}(\mathtt{P3})\}$, $\mathcal{S}(\mathtt{P1}) = \{R^1(\cdot,\cdot)\}$, $\mathcal{S}(\mathtt{P2}) = \{R^2(\cdot,\cdot)\}$, $\mathcal{S}(\mathtt{P3}) = \{R^3(\cdot,\cdot)\}$.

The local instances are: $D(\mathtt{P1}) = \{\}$, $D(\mathtt{P2}) = \{R^2(a,b)\}$, $D(\mathtt{P3}) = \{R^3(a,c)\}$. The DECs are:

1. $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall x \forall y (R^2(x,y) \rightarrow R^1(x,y))\}$,
2. $\Sigma(\mathtt{P1}, \mathtt{P3}) = \{\forall x \forall y (R^3(x,y) \rightarrow R^1(x,y))\}$, and
3. $\Sigma(\mathtt{P1}, \mathtt{P1}) = \{\forall x \forall y \forall z (R^1(x,y) \wedge R^1(x,z) \rightarrow y = z)\}$.

$\Sigma(\mathtt{P1}, \mathtt{P2})$ and $\Sigma(\mathtt{P1}, \mathtt{P3})$ contain import DECs. However, due to the local constraints in $\Sigma(\mathtt{P1}, \mathtt{P1})$, there are no solutions for P1. □

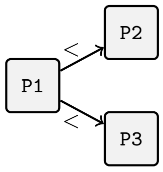


Fig. 7. Accessibility graph for Example 5.8

Example 5.9
Consider $D(\texttt{P}) = \{P(a,b), P(a,c)\}, D(\texttt{Q}) = \{Q(a,d)\}$, and $\mathit{Trust} = \{(\texttt{P}, \mathit{less}, \texttt{Q})\}$. $\Sigma(\texttt{P},\texttt{Q}) = \{\forall x \forall y (Q(x,y) \rightarrow P(x,y)\}$, and $\Sigma(\texttt{P},\texttt{P}) = \{\forall x \forall y \forall z \forall v (P(x,y) \wedge P(x,z) \wedge P(x,v) \rightarrow y = z \vee z = v \vee v = y)\}$. The local constraints in $\Sigma(\texttt{P},\texttt{P})$ ensures that there are at most two tuples in P with the same first attribute. In this case, $\Sigma(\texttt{P},\texttt{Q})$ is of the import kind and therefore $P(a,d)$ will belong to all solutions. The local constraint in $\Sigma(\texttt{P},\texttt{P})$ will force the removal of either $P(a,b)$ or $P(a,c)$, and therefore P has two solutions: $\{P(a,d), P(a,b)\}$ and $\{P(a,d), P(a,c)\}$. □

6 Answer Set Programs and the Solutions for a Peer

Continuing with the special PDES semantics based on the use of *null* introduced in Section 4, in this section we show how to specify neighborhood solutions for a peer (which is the central notion in this paper, upon which the recursive notion of global solution is defined) by means of disjunctive logic programs with stable model semantics (Gelfond and Lifschitz 1991; Eiter et al. 1997). As above, we assume that DECs and ICs have the forms (1) or (2).

Our aim in this section is to show the gist of the approach, by considering in precise terms a particular but still interesting case, one where the RDECs have a limited syntactic form, without joins in the consequents. The general case, i.e. with arbitrary UDECs and RDECs as considered in this work so far, is possible and not difficult from the conceptual or technical point of view, but more difficult or lengthy to present.

Remark 6.1
In this section we make the additional assumption that given a schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$, for each $\texttt{P} \in \mathcal{P}$, its set of DECs $\Sigma(\texttt{P})$ is *ref-acyclic*. In this case, we say that $\mathfrak{P}$ is ref-acyclic. The formulation of the programs is independent from this assumption, but -as we will see later in this section- it ensures desirable properties of the program. □

Now we show how to specify the neighborhood solutions for a peer P as the stable models of a disjunctive logic program $\Pi(\texttt{P})$. *The program can be used for this purpose under the assumption that* P *has already gathered, for each of its neighbors* Q, *the intersection of its solutions*, i.e. $\mathit{Core}(\texttt{Q}, \mathfrak{D})$ (cf. Definition 5.6). However, the program works with arbitrary instances for the neighbors. Accordingly, the facts of the program come from the union $\mathcal{D}$ of the given instances for the peers in $\mathcal{N}(\texttt{P})$, the neighborhood centered around peer P. As a consequence, $\Pi(\texttt{P})$ can be used to compute the restrictions to P'schema of P's neighborhood solutions, on the basis of a neighborhood instance $\mathcal{D}$.

In $\Pi(\texttt{P})$, we find each predicate $R \in \mathcal{S}(\mathcal{N}(\texttt{P}))$ and also its copy $R\_$ that has an extra argument (an augmented *nickname*). This argument is the last one and is used to place an annotation constant. Thus, if R has arity n, then $R\_$ has arity $n+1$. The possible annotation constants are: $\mathbf{t}, \mathbf{f}, \mathbf{t}^\star, \mathbf{f}^\star, \mathbf{t}^{\star\star}$. Their occurrences in database tuples have the following intended semantics:

Annotation	The tuple $R(\bar{a})$ ...
$R\_(\bar{a}, \mathbf{t})$	is made true in (inserted into) the database
$R\_(\bar{a}, \mathbf{f})$	is made false in (deleted from) the database
$R\_(\bar{a}, \mathbf{t}^\star)$	was true or is made true
$R\_(\bar{a}, \mathbf{f}^\star)$	was false or is made false
$R\_(\bar{a}, \mathbf{t}^{\star\star})$	was true or made true, and is not deleted

This intended semantics is formally captured in the program by means of appropriate rules. The (possibly virtual) insertions and deletions are made in order to satisfy P's DECs. Actually, for each DEC 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 6.1 for UDECs, and 4. and 5. for RDECs). That is why we use annotations $\mathbf{t}$ or $\mathbf{f}$ in rule heads.

The annotations $\mathbf{t}^\star, \mathbf{f}^\star$ are used to execute sequences of virtual updates, which may be necessary when there are interacting DECs. 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).

Before presenting the logic program, let us recall that $RelV(\psi)$ denotes the set of relevant variables of a constraint ψ, those where the occurrence of *null* is relevant for its satisfaction (see Definition 4.5). Those attributes/variables receive a special treatment in the program.

In the following, for a DEC ψ, $A(\psi)$ and $C(\psi)$ denote the set of database atoms (without built-ins) in the antecedent of, resp. the consequent, of ψ.

A consequence of repairing using *null*, is that any RDEC with joins between existential quantifiers can only be repaired by deleting tuples and not by inserting tuples with *null*. Thus, the most relevant RDECs in this setting are of the form:

$$\forall \bar{x}(R(\bar{x}) \longrightarrow \exists \bar{y}\, Q(\bar{x}', \bar{y})), \tag{34}$$

For this reason, and to simplify the presentation, the logic program that follows considers RDECs of this form only.

Definition 6.1
Consider a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, Trust \rangle$ and a peer $\texttt{P} \in \mathcal{P}$ with local instance $D(\texttt{P})$ (for the schema $\mathcal{S}(\texttt{P})$). Let $\bar{D}$ be an instance for the schema $\mathcal{S}(\mathcal{N}(\texttt{P}))$, i.e. for P's neighborhood $\mathcal{N}(\texttt{P})$, with $\bar{D}\lfloor\mathcal{S}(\texttt{P}) = D(\texttt{P})$. The *solution program* $\Pi(\texttt{P}; \bar{D})$ for P contains:

1. The following facts: $dom(a)$, for every $a \in Adom(\bar{D})$; $dom(null)$, and $R(\bar{a})$, for every atom $R(\bar{a}) \in \bar{D}$.[21]

2. For every UDEC $\psi \in \Sigma(\mathtt{P},\mathtt{Q})$ of the form (1), with $\mathtt{Q} \in \mathcal{N}(\mathtt{P})$ and $(\mathtt{P}, less, \mathtt{Q}) \in Trust$, the rule:

$$\bigvee\{R\_(\bar{x}_i,\mathbf{f}) \mid R(\bar{x}_i) \in A(\psi), R \in \mathcal{S}(\mathtt{P})\} \vee \bigvee\{Q\_(\bar{y}_j,\mathbf{t}) \mid Q(\bar{y}_j) \in C(\psi), Q \in \mathcal{S}(\mathtt{P})\}$$
$$\leftarrow \bigwedge_{i=1}^{n} R_i\_(\bar{x}_i,\mathbf{t}^\star),\ \bigwedge_{j=1}^{m} Q_j\_(\bar{y}_j,\mathbf{f}^\star),\ \bigwedge_{x_l \in RelV(\psi)} x_l \neq null\cdot$$

Here, when $Q_j \in \mathcal{B}$ (i.e. a built-in), $Q_j\_(\bar{y}_j,\mathbf{f}^\star)$ simply denotes the negated atom $Q_j\_(\bar{y}_j)$, e.g. $=$ becomes $\neq$, $IsNull$ becomes, $IsNotNull$, etc., without annotations. Notice that the condition $Q \in \mathcal{S}(\mathtt{P})$ in the head of the rule discards built-in atoms (cf. Example 6.1 below).

3. For every UDEC $\psi \in \Sigma(\mathtt{P},\mathtt{Q})$ of the form (1), with $\mathtt{Q} \in \mathcal{N}(\mathtt{P})$ and $(\mathtt{P}, same, \mathtt{Q}) \in Trust$, the rule:

$$\bigvee\{R\_(\bar{x}_i,\mathbf{f}) \mid R(\bar{x}_i) \in A(\psi)\} \vee \bigvee\{Q\_(\bar{y}_j,\mathbf{t}) \mid Q(\bar{y}_j) \in C(\psi)\} \leftarrow \bigwedge_{i=1}^{n} R_i\_(\bar{x}_i,\mathbf{t}^\star),$$
$$\bigwedge_{j=1}^{m} Q_j\_(\bar{y}_j,\mathbf{f}^\star), \bigwedge_{x_l \in RelV(\psi)} x_l \neq null.$$

We apply the same convention as in 2. about the rule body. This case also covers UDECs in $\Sigma(\mathtt{P},\mathtt{P})$, i.e. local universal ICs for $\mathtt{P}$.

4. For every RDEC $\psi \in \Sigma(\mathtt{P},\mathtt{Q})$ of the form (34), with $\mathtt{Q} \in \mathcal{N}(\mathtt{P})$ and $(\mathtt{P}, same, \mathtt{Q}) \in Trust$, the rule:

$$R\_(\bar{x},\mathbf{f}) \vee Q\_(\bar{x}', \overline{null}, \mathbf{t}) \leftarrow R\_(\bar{x},\mathbf{t}^\star), not\ aux_\psi(\bar{x}'), \bar{x}' \neq null\cdot$$

Here, aux_ψ is an auxiliary predicate used to obtain a *safe* rule (i.e. its variables in negated atoms also appear in a positive atom), whose body captures the violations of (34). In FO logic this would be done through: $R(\bar{x}) \wedge \neg\exists\bar{y}\, Q(\bar{x}',\bar{y})$, which basically leads to the following rules that define the auxiliary predicate (taking into account the special semantics of *null*):

For every $y_i \in \bar{y}$:

$$aux_\psi(\bar{x}') \leftarrow Q\_(\bar{x}',\bar{y},\mathbf{t}^\star), not\, Q\_(\bar{x}',\bar{y},\mathbf{f}), \bar{x}' \neq null, y_i \neq null\cdot \quad (35)$$

$$aux_\psi(\bar{x}') \leftarrow Q(\bar{x}',\overline{null}), not\, Q\_(\bar{x}',\overline{null},\mathbf{f}), \bar{x}' \neq null\cdot \quad (36)$$

This case covers the RDECs in $\Sigma(\mathtt{P},\mathtt{P})$, i.e. local referential ICs for $\mathtt{P}$. (Cf. Example 6.1 below.) Here, and in the rest of this work, atoms of the form $Q\_(\bar{x}',\overline{null},\ldots)$ associated to an atom in a DEC, have all the existential variables appearing in the latter and predicate Q replaced by *null*. Furthermore, for $\bar{x} = x_1,\ldots,x_n$, $\bar{x} \neq null$ abbreviates $x_1 \neq null,\ldots,x_n \neq null$.

5. For every RDEC $\psi \in \Sigma(\mathtt{P},\mathtt{Q})$ of the form (34), with $\mathtt{Q} \in \mathcal{N}(\mathtt{P})$ and $(\mathtt{P}, less, \mathtt{Q}) \in Trust$, the rules:

(a) If $R \in \mathcal{S}(\mathtt{P})$: $R\_(\bar{x},\mathbf{f}) \leftarrow R\_(\bar{x},\mathbf{t}^\star), not\ aux_\psi(\bar{x}'), \bar{x}' \neq null\cdot$
(b) If $Q \in \mathcal{S}(\mathtt{P})$: $Q\_(\bar{x}',\overline{null},\mathbf{t}) \leftarrow R\_(\bar{x},\mathbf{t}^\star), not\ aux_\psi(\bar{x}'), \bar{x}' \neq null\cdot$

Plus rules (36), (35).

6. For each predicate $R \in \mathcal{S}(\mathcal{N}(\mathtt{P}))$, the annotation rules:

[21] We can also add $IsNull(null)$, and $IsNotNull(c)$, for every non-null constant in the active domain, but this is not necessary if these two predicates do not appear in the program.

$R\_(\bar{x}, \mathbf{f}^\star) \leftarrow \mathit{dom}(\bar{x}), \mathit{not}\, R(\bar{x})\cdot$ $\qquad R\_(\bar{x}, \mathbf{f}^\star) \leftarrow R\_(\bar{x}, \mathbf{f})\cdot$

$R\_(\bar{x}, \mathbf{t}^\star) \leftarrow R(\bar{x})\cdot$ $\qquad R\_(\bar{x}, \mathbf{t}^\star) \leftarrow R\_(\bar{x}, \mathbf{t})\cdot$

7. For each predicate $R \in \mathcal{S}(\mathcal{N}(\mathtt{P}))$, the program constraint:
   $\leftarrow R\_(\bar{x}, \mathbf{t}),\ R\_(\bar{x}, \mathbf{f})\cdot$
8. For each predicate $R \in \mathcal{S}(\mathtt{P})$, the interpretation rule:
   $R\_(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow R\_(\bar{x}, \mathbf{t}^\star),\ \mathit{not}\, R\_(\bar{x}, \mathbf{f})\cdot$ □

Example 6.1
If the UDEC ψ is $\forall x \forall y (R(x) \ \rightarrow \ S(x,y) \vee \mathit{IsNull}(y))$, then the corresponding rule according to 2. above is: $R(x, \mathbf{f}) \leftarrow R(x, \mathbf{t}^\star), S(x, y, \mathbf{f}^\star), \mathit{IsNotNull}(y), x \neq \mathit{null}$.

Notice that $\mathit{RelV}(\psi) = \{x\}$. Since the program interprets *null* as any other constant in the domain, this rule can be replaced by $R(x, \mathbf{f}) \leftarrow R(x, \mathbf{t}^\star), S(x, y, \mathbf{f}^\star), y \neq \mathit{null}, x \neq \mathit{null}$.

If the RDEC ψ is $\forall x (R(x) \rightarrow \exists y S(x,y))$, then the corresponding rules according to 4. above are:

$$\begin{array}{rcl} R\_(x, \mathbf{f}) \vee S\_(x, \mathit{null}, \mathbf{t}) & \leftarrow & R(x, \mathbf{t}^\star), \mathit{not}\ \mathit{aux}(x), x \neq \mathit{null} \cdot \\ \mathit{aux}(x) & \leftarrow & S(x, \mathit{null}), \mathit{not}\, S\_(x, \mathit{null}, \mathbf{f}), x \neq \mathit{null} \cdot \\ \mathit{aux}(x) & \leftarrow & S\_(x, y, \mathbf{t}^\star), \mathit{not}\, S\_(x, y, \mathbf{f}), x \neq \mathit{null}, y \neq \mathit{null} \cdot \end{array}$$

In this case, the RDEC is "repaired", as usual with referential ICs in databases, by either deleting the tuple from table R or inserting a tuple with a null value in table S, both cases equally acceptable. The two cases for item 5. in the program correspond each to only one of these possible choices. □

Since instance $\bar{D}$ and its active domain are finite, the program has a finite number of facts in item 1. The most relevant part of the program corresponds to the rules 2.-5. They capture through their bodies the violations of DECs; and through their heads, the (possibly alternative) virtual updates that are necessary on the peers' instances to restore the satisfaction of DECs and local ICs. In the bodies of these 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 special semantics of null values introduced in Section 4.

The fixed semantics of annotations is captured by the rules 6.-8. They appear in every solution program. The program constraints in 7. discard models where an atom is both inserted and deleted.

The instances $\bar{D}\!\downharpoonright\!\mathcal{S}(\mathtt{Q}))$ used in the program for P's neighbors may not coincide with the initial, physical instances $D(\mathtt{Q})$ in an instance $\mathfrak{D}$ for the PDES $\mathfrak{P}$ (except when Q is P). However, the idea is that P, given a global instance $\mathfrak{D}$, uses its program with $D(\mathtt{P})$ and $\bar{D}\!\downharpoonright\!\mathcal{S}(\mathtt{Q})) = \mathit{Core}(\mathtt{Q}, \mathfrak{D})$ for each $\mathtt{Q} \in \mathcal{N}^\circ(\mathtt{P})$. In this way, the program computes through its stable models the neighborhood solutions for P, whose restrictions to P's schema will be the solutions for P. Notice that atoms annotated with $\mathbf{t}^{\star\star}$ in a stable model of P's program have predicates in $\mathcal{S}(\mathtt{P})$ only; and they define a database instance for P.

The adoption of the stable model semantics for the solution programs (Gelfond and Lifschitz 1991), plus the appropriate use of conditions with *null* in rules' bodies, guarantee the minimal discrepancy between the generated and the original instances according to Definitions 3.1 and 5.3.

Example 6.2
Consider $\mathcal{P} = \{\mathtt{P1}, \mathtt{P2}\}$, with $D(\mathtt{P1}) = \{R^1(a, 2)\}$, $D(\mathtt{P2}) = \{R^2(d, 5)\}$, and neighborhood instance $\bar{D} = D(\mathtt{P1}) \cup D(\mathtt{P2})$. Assume $(\mathtt{P1}, \mathit{less}, \mathtt{P2}) \in \mathit{Trust}$, and the only set of DECs in the system is $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall xy(R^2(x, y) \rightarrow R^1(x, y))\}$.

The solution program $\Pi(\mathtt{P1}; \bar{D})$ contains: (we itemize in correspondence to Definition 6.1)

1. The facts $\mathit{dom}(a), \mathit{dom}(d), \mathit{dom}(2), \mathit{dom}(5), \mathit{dom}(\mathit{null}), R^1(a, 2), R^2(d, 5)\cdot$
2. $R^1\_(x, y, \mathbf{t}) \leftarrow R^2\_(x, y, \mathbf{t}^\star),\ R^1\_(x, y, \mathbf{f}^\star), x \neq \mathit{null}, y \neq \mathit{null}\cdot$
6. $R^1\_(x, y, \mathbf{f}^\star) \leftarrow R^1\_(x, y, \mathbf{f})\cdot \qquad R^1\_(x, y, \mathbf{f}^\star) \leftarrow \mathit{dom}(x), \mathit{dom}(y), \mathit{not}\, R^1(x, y)\cdot$
   $R^1\_(x, y, \mathbf{t}^\star) \leftarrow R^1\_(x, y, \mathbf{t})\cdot \qquad R^1\_(x, y, \mathbf{t}^\star) \leftarrow R^1(x, y)\cdot$
7. $\leftarrow R^1\_(x, y, \mathbf{t}), R^1\_(x, y, \mathbf{f})\cdot$
8. $R^1\_(x, y, \mathbf{t}^{\star\star}) \leftarrow R^1\_(x, y, \mathbf{t}^\star), \mathit{not}\, R^1\_(x, y, \mathbf{f})\cdot$

The rule in 2. makes sure that a *null*-free R^2-tuple that is not in R^1, is also virtually inserted into R^1 (if it had *null*, the DEC would be satisfied since all attributes are relevant). Since P1 trusts P2 more than itself, virtual changes affect only peer P1. According to Definition 6.1, we should also include rules similar to those in 6. and 7. for R^2. However, they are not necessary because R^2 does not change. If we decide to omit them, we have to replace the atom $R^2\_(x, y, \mathbf{t}^\star)$ in the body of rule in 2. by $R^2(x, y)$.

Notice that since P2, which is a sink peer, does not have local ICs, its instance $D(\mathtt{P2})$ is also its only solution, and the one passed over to P1, who uses it to build the facts of its program. □

Example 6.3
Consider $\mathcal{P} = \{\mathtt{P1}, \mathtt{P2}\}$, with $D(\mathtt{P1}) = \{R^1(s, t),\ R^1(a, \mathit{null})\}$, $D(\mathtt{P2}) = \{R^2(c, d), R^2(a, e)\}$, and $\mathit{Trust} = \{(\mathtt{P1}, \mathit{same}, \mathtt{P2})\}$.

Assume the only set of DECs are $\Sigma(\mathtt{P1}, \mathtt{P1}) = \{\forall xyz(R^1(x, y) \wedge R^1(x, z) \rightarrow y = z)\}$, and $\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall xy(R^2(x, y) \rightarrow \exists z R^1(x, z))\}$. The neighborhood instance is $\bar{D} = D(\mathtt{P1}) \cup D(\mathtt{P2})$ (the second disjunct is also P2's only solution).

The solution program $\Pi(\mathtt{P1}; \bar{D})$ for P1 is as follows: (omitting rules 6. and 7.):

1. $\mathit{dom}(a), \ldots, \mathit{dom}(\mathit{null}), R^1(a, \mathit{null}), R^1(s, t), R^2(c, d), R^2(a, e)\cdot$
3. $R^1(x, y, \mathbf{f}) \vee R^1(x, z, \mathbf{f}) \leftarrow R^1(x, y, \mathbf{t}^\star), R^1(x, z, \mathbf{t}^\star), x \neq \mathit{null}, y \neq \mathit{null},$
   $z \neq \mathit{null},\ y \neq z\cdot$
4. $R^2\_(x, y, \mathbf{f}) \vee R^1\_(x, \mathit{null}, \mathbf{t}) \leftarrow R^2\_(x, y, \mathbf{t}^\star), \mathit{not}\ \mathit{aux}(x), x \neq \mathit{null}\cdot$
   $\mathit{aux}(x) \leftarrow R^1(x, \mathit{null}), \mathit{not}\, R^1\_(x, \mathit{null}, \mathbf{f})\cdot$
   $\mathit{aux}(x) \leftarrow R^1(x, y, \mathbf{t}^\star), \mathit{not}\, R^1\_(x, y, \mathbf{f}), x \neq \mathit{null}, y \neq \mathit{null}\cdot$
8. $R^1\_(x, y, \mathbf{t}^{\star\star}) \leftarrow R^1\_(x, y, \mathbf{t}^\star), \mathit{not}\, R^1\_(x, y, \mathbf{f})\cdot$

Rule 3. takes care of P1's the local functional dependency. In case of a violation, one of the two tuples in conflict has to be deleted. Rule 4. has the role of satisfying the RDEC. Since `P1` trusts `P2` the same as itself, consistency is restored by either deleting the tuple from R^2 or introducing a *null* into R^1.

Here, only $R^1\_$ -atoms become annotated with $\mathbf{t}^{\star\star}$. These annotations are used to build solutions for `P`. In this example, we have two stable models, whose restrictions to $\mathbf{t}^{\star\star}$-annotated atoms are: $\{R^1\_(a, null, \mathbf{t}^{\star\star}), R^1\_(s, t, \mathbf{t}^{\star\star}), R^1\_(c, null, \mathbf{t}^{\star\star})\}$, corresponding to the insertion of tuple $R^1(c, null)$; and $\{R^1\_(a, null, \mathbf{t}^{\star\star})$, $R^1\_(s, t, \mathbf{t}^{\star\star})\}$, corresponding to the deletion of tuple $R^2(c, d)$. They correspond to the two solutions for `P`: $\{R^1(a, null), R^1(s, t), R^1(c, null)\}$ and $\{R^1(a, null), R^1(s, t)\}$. □

Notice that each peer `P` in a PDES has a solution program that, except for the facts in it, is fixed. It depends only on its DECS in Σ(`P`) and the trust relationships to its neighbors. The same program can be used with different initial neighborhood instances $\bar{D}$. On their basis, the program will compute stable models, whose double-starred atoms will determine a local instance.

Definition 6.2
Consider a PDES $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, Trust\rangle$, a peer $\texttt{P} \in \mathcal{P}$, and a neighborhood instance $\bar{D}$ for $\mathcal{S}(\mathcal{N}(\texttt{P}))$. Let M be a stable model of $\Pi(\texttt{P}; \bar{D})$. The database instance for peer `P` (and schema $\mathcal{S}(\texttt{P})$) associated to M is $D_M = \{R(\bar{a}) \mid R \in \mathcal{S}(\texttt{P}) \text{ and } R\_(\bar{a}, \mathbf{t}^{\star\star}) \in M\}$. □

The following result tells us that the instances for a peer obtained via the stable models of its program are all and only the solutions for the peer, under the assumption that its neighbors contribute with their own cores.

Proposition 6.1
Consider a PDES $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, Trust\rangle$, with an instance $\mathfrak{D}$ for $\mathfrak{P}$, and $\texttt{P} \in \mathcal{P}$ whose set of DECs $\Sigma(\texttt{P})$ is ref-acyclic. Given the neighborhood instance for $\mathcal{S}(\mathcal{N}(\texttt{P}))$: $\bar{D} := D(\texttt{P}) \cup \bigcup_{\texttt{Q} \in \mathcal{N}^\circ(\texttt{P})} Core(\texttt{Q}, \mathfrak{D})$, it holds:

$$Sol(\texttt{P}, \mathfrak{D}) = \{D_M \mid M \text{ is a stable model of program } \Pi(\texttt{P}, \bar{D})\}. \qquad \square$$

This result generalizes one about the correctness of similar programs for null-based repairs of the kind considered in this work, for single databases with respect to denial constraints and referential constraints and null values (Bravo and Bertossi 2006). The hypothesis of ref-acyclicity is necessary for the soundness of the program (every instance D_M is a solution). Otherwise, only the completeness of the program can be guaranteed (every solution is a D_M) (Bravo and Bertossi 2006).

Under the assumption that we have already computed the (intersection of the) solution instances for `P`'s neighbors, the program for `P` allows us to compute its solution instances. This generates a recursive process that can be applied because $\mathcal{G}(\texttt{P})$ is acyclic. Terminal peers `P'` in $\mathcal{G}(\texttt{P})$, i.e. without outgoing edges, become the base cases for the recursion. If their local instances $D(\texttt{P'})$ are (locally) consistent, they pass those instances to their neighbors. Otherwise, they first, using program $\Pi(\texttt{P'}; D(\texttt{P'}))$ compute local repairs for $D(\texttt{P'})$, and pass the intersection of them to their neighbors.

Several optimizations can be applied to solution programs (Caniupan and Bertossi 2010). An important one has to do with the materialization of the *closed-world-assumption*, which results through the rule $R^1\_(x, y, \mathbf{f}^\star) \leftarrow dom(x), dom(y)$, *not* $R^1(x,y)$ in 6. in Example 6.2. This is clearly undesirable and can be avoided. We do not provide optimized versions here, because they are more difficult to read.

Proposition 6.1 still holds if P, instead of collecting the intersection of the solutions of a neighbor Q, uses the intersection of the solutions for Q restricted to the subschema of Q that contains Q's relations that appear in $\Sigma(\mathtt{P},\mathtt{Q})$, which are those P needs to run its program.

As we expressed at the beginning of this section, our solution programs can be much more general. In particular, they can be modified to include rules for REDCs with existential quantifiers and joins in the consequents. For example, if the DEC is, say $\forall xy(R(x,y) \wedge S(y,z) \rightarrow \exists w(P(x,w) \wedge Q(w,z)))$, and the antecedent is true with *null*-free tuples,[22] but not the consequent, one of the (satisfying) tuples in the antecedent has to be deleted. All we need to handle this DEC is a disjunctive, "deleting" rule. Built-in comparisons (with *null*) are used by the rule as well.

Remark 6.2
For simplicity, the programs we introduced above do not consider inconsistent peers, i.e. a peer P whose instance is $D^{\mathtt{P}} = \{\mathbf{inc}_{\mathtt{P}}\}$. As the following example shows, it is easy to modify the rules to capture this situation. This change is compatible with the corresponding general DECs (cf. Remark 3.1). □

Example 6.4
(example 6.2 continued) The system is exactly as before, except that now $D^{\mathtt{P2}} = \bigcap Sol(\mathtt{P2},\mathfrak{D}) = \{\mathbf{inc}_{\mathtt{P2}}\}$, reflecting the fact that P2 has no solutions (due to its mappings and trust relationships to other peers, which we are not showing here). The general program that allows for inconsistent peers would now have, instead of 1. and 2.:

1$'$ $\cdot$ $dom(a), dom(2), R^1(a,2), \mathbf{inc}_{\mathtt{P2}}\cdot$

2'. $R^1\_(x,y,\mathbf{t}) \leftarrow R^2\_(x,y,\mathbf{t}^\star),\ R^1\_(x,y,\mathbf{f}^\star), not\ \mathbf{inc}_{\mathtt{P2}},\ x \neq null, y \neq null\cdot$

In this case, and as expected, the rule has no effect on P1. □

6.1 ASPs and PCAs

With a solution program for P, PCAs to a query $\mathcal{Q}$ posed to P can be obtained by running a query program in combination with the solution program. First a query program $\Pi(\mathcal{Q})$ has to be produced, which is rather standard, and next, $\Pi(\mathtt{P}) \cup \Pi(\mathcal{Q})$ is run under the *cautious stable model semantics*, the one that declares as true what is simultaneously true in all the stable models. Of course, the same program $\Pi(\mathtt{P})$ can be used with different queries.

Example 6.5
(example 6.2 continued) In order to obtain P1's PCAs to the query $\mathcal{Q}_1(x,y) : R^1(x,y)$, the rule $Ans_1(x,y) \leftarrow R^1(x,y,\mathbf{t}^{\star\star})$ has to be added to $\Pi(\mathtt{P}; D(\mathtt{P1}) \cup D(\mathtt{P2}))$. The PCAs are the ground Ans_1-atoms in the intersection of all stable

[22] If it is true via tuples with nulls, then due to the relevance of attributes, the DEC is immediately satisfied.

models of the combined program. For the query $\mathcal{Q}_2(x)\colon \exists y R^1(x,y)$, the query rule is: $Ans_2(x) \leftarrow R^1(x,y,\mathbf{t}^{\star\star})$. □

In general, given a conjunctive query $\mathcal{Q}$ (for which we want PCAs), first its rewriting $\mathcal{Q}^N$ is produced (as in Definition 4.6). Next, $\mathcal{Q}^N$ is rewritten in its turn as a query program with annotation $\mathbf{t}^{\star\star}$. The query program is added to the peer's solution program, and the combination is run under the skeptical semantics.

Deciding the truth of ground atomic queries by means of the solution program for a peer under the skeptical semantics amounts to deciding membership of the local core, which, by Corollary 5.1, is Π_2^P-complete in data. This is exactly the data complexity of deciding skeptical entailment for atomic queries for disjunctive logic programs under the stable model semantics (Dantsin et al. 2001). So, the solution programs have the right expressive power for the problem at hand.

Using solution programs, our semantics could be naively implemented as follows. When $\mathtt{P}$ is posed a query, $\mathtt{P}$ has to run its program, for which it needs as facts those in the intersections of the solutions of its neighbors. So, $\mathtt{P}$ sends to each neighbor $\mathtt{Q}$ queries of the form $\mathcal{Q}\colon R(\bar{x})$, where $R \in \mathcal{S}(\mathtt{Q})$ and appears in $\Sigma(\mathtt{P},\mathtt{Q})$. Peer $\mathtt{P}$ expects to receive from $\mathtt{Q}$ the PCAs to $\mathcal{Q}$, because they corresponds to the extension of R in the intersection of solutions for $\mathtt{Q}$.

In order to return to $\mathtt{P}$ the PCAs to its queries, each of its neighbors $\mathtt{Q}$ has to run its own program $\Pi(\mathtt{Q})$. As before, they need PCAs from their own neighbors, etc. This recursion eventually reaches peers that have no DECs, and assuming its local ICs are satisfied, they return query answers to its neighbors directly from their original instances. This is the start of the backward propagation of PCAs process, which goes on until reaching $\mathtt{P}$. Eventually, $\mathtt{P}$ gets all the facts to run its program and obtain the PCAs to the original query. If the local instance of a terminal peer is not consistent with respect to its local ICs, the local solution program can still be used to restore consistency with respect to the local ICs, as in consistent query answering. The data to be sent back to the neighbors comes from the intersection of the repairs, in the form of consistent query answers (Arenas et al. 1999).

Example 6.6
(example 2.1 continued) Consider an instance $\mathfrak{D} = \{D(\mathtt{P1}), D(\mathtt{P2}), D(\mathtt{P3})\}$ with $D(\mathtt{P1}) = \{R^1(a,2)\}$, $D(\mathtt{P2}) = \{\ R^2(c,4), R^2(d,5)\}$, and $D(\mathtt{P3}) = \{R^3(c,4)\}$. A user poses the query $\mathcal{Q}_0\colon R^1(x,y)$ to $\mathtt{P1}$, expecting its PCAs.

To run its program, $\mathtt{P1}$ needs the intersection of the solutions of peer $\mathtt{P2}$. So, $\mathtt{P1}$ sends to $\mathtt{P2}$ the queries $\mathcal{Q}_1^1\colon R^2(x,y)$ and $\mathcal{Q}_2^1\colon S^2(x,y)$ (actually, $\mathtt{P1}$ does not need the latter, S^2 is not relevant to $\mathtt{P1}$).

In order to peer-consistently answer these queries, $\mathtt{P2}$ needs from $\mathtt{P3}$ the PCAs to $\mathcal{Q}_3^2$: $R^3(x,y)$. Since $\mathtt{P3}$ has no neighbors, it returns to $\mathtt{P2}$ the entire extension in its local database, i.e. $D(\mathtt{P3}) = \{R^3(c,4)\}$. Now, $\mathtt{P2}$ runs its program $\Pi(\mathtt{P2}; D(\mathtt{P2}) \cup D(3))$. It contains, among others, the facts $R^2(c,4), R^2(d,5), R^3(c,4)$; and (assuming $\Sigma(\mathtt{P2},\mathtt{P2}) = \emptyset$) the main rule:

$$R^2\_(x,y,\mathbf{f}) \vee R^3\_(x,y,\mathbf{f}) \leftarrow R^2\_(x,y,\mathbf{t}^{\star}), R^3\_(x,y,\mathbf{t}^{\star}), x \neq null, y \neq null\cdot$$

$\mathtt{P2}$ has two solutions, $\{R^2(d,5)\}$ and $\{R^2(c,4), R^2(d,5)\}$, whose intersection, $Core(\mathtt{P2},\mathfrak{D}) = \{R^2(d,5)\}$, is given to $\mathtt{P1}$.

Finally, the program $\Pi(\mathtt{P1}; D(\mathtt{P1}) \cup Core(\mathtt{P2}, \mathfrak{D})$ (cf. Example 6.2) is run. It has only one solution, namely $\{R^1(a,2),\ R^1(d,5)\}\}$; and the PCAs to $\mathcal{Q}_0$ are $\langle a, 2\rangle$ and $\langle d, 5\rangle$. □

6.2 The import case revisited

We first consider the unrestricted case introduced in Section 5.5.1. Here, each peer $\mathtt{P}$ has a solution program $\Pi^-(\mathtt{P})$ as in Definition 6.1, with rules of the form:

2. $S\_(\bar{y}_j, \mathbf{t}) \ \leftarrow\ \bigwedge_{i=1}^n R_i\_(\bar{x}_i, \mathbf{t}^\star),\ S\_(\bar{y}_j, \mathbf{f}^\star),\ \bigwedge_{x_l \in RelV(\psi)} x_l \neq null,\ \bar{\varphi}\cdot$

Here, $S \in \mathcal{S}(\mathtt{P})$ and the $R_i \in \mathcal{S}(\mathtt{Q})$, for $\mathtt{Q} \in \mathcal{N}^\circ(\mathtt{P})$.

5.(b) $Q\_(\bar{x}', \overline{null}, \mathbf{t}) \leftarrow R\_(\bar{x}, \mathbf{t}^\star),\ not\ aux_\psi(\bar{x}'),\ \bar{x}' \neq null\cdot$

Here, $Q \in \mathcal{S}(\mathtt{P})$ and $R \in \mathcal{S}(\mathtt{Q})$, for $\mathtt{Q} \in \mathcal{N}^\circ(\mathtt{P})$.

We also need the auxiliary rules (36), (35), and only for predicates in $\mathcal{S}(\mathtt{P})$ that appear in consequents of RDECs.

Now, for a predicate $Q \in \mathcal{S}(\mathtt{P})$, a solution program like this never generates a tuple of the form $Q(\bar{a}, \mathbf{t})$. In consequence, the negative literals can be eliminated from the bodies of both (36) and (35). The resulting program still contains negation, in rules 6. and 8. However, the solution program becomes a (non-disjunctive) stratified normal program (Abiteboul et al. 1995). In consequence, for every set of facts, it has only one stable model that can be computed in polynomial time in data. It corresponds to the only solution for peer $\mathtt{P}$ mentioned in Section 6.1.

If the query $\mathcal{Q}$ (or, better, its rewriting $\mathcal{Q}^N$) posed to $\mathtt{P}$ can also be represented as a stratified normal program $\Pi(\mathcal{Q})$ (which is the case when $\mathcal{Q}$ is first-order, for example), then the combined program $\Pi(\mathtt{P}) \cup \Pi(\mathcal{Q})$ is also normal and stratified. In consequence, computing PCAs can also be done in polynomial time. This is under the assumption that the peer has collected, for each of its neighbors, the intersection of its (local) solutions. We have obtained the following result.

Proposition 6.2
For an unrestricted import PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, Trust\rangle$, an instance $\mathfrak{D}$ for $\mathfrak{P}$, and a peer $\mathtt{P} \in \mathcal{P}$, deciding answers to a conjunctive query posed to $\mathtt{P}$ that are true in $\bigcap NS(\mathtt{P}, \bar{D})$, where $\bar{D} = D(\mathtt{P}) \cup \bigcup_{\mathtt{Q} \in \mathcal{N}^\circ(\mathtt{P})} Core(\mathtt{Q}, \mathfrak{D})$, can be done in polynomial time in the size of $\bar{D}$. □

In the restricted import case of Section 5.5.2, we allowed peers to have local ICs. In this case, as illustrated in Examples 5.8 and 5.9 the situation may change: Due to the local ICs, the solution program may be disjunctive, and have none or several stable models, which is also reflected in the peers' solutions.

Example 6.7
(example 5.8 continued) In this case we have no solution. This is captured by the corresponding program through the program constraint $\leftarrow R^1\_(x, y, \mathbf{t}), R^1\_(x, y, \mathbf{f})$. In each of the two models of the program without the constraint, there will be the atoms $R^1(a, b, \mathbf{t}), R^1(a, c, \mathbf{t}), R^1(a, b, \mathbf{f})$ or the atoms $R^1(a, b, \mathbf{t}), R^1(a, c, \mathbf{t}), R^1(a, c, \mathbf{f})$. Both models will be discarded for violating the constraint. □

Example 6.8
(example 5.9 continued) The solution program for P has as main rules, the following:

$$P\_(x, y, \mathbf{t}) \leftarrow Q\_(x, y, \mathbf{t}^\star), P\_(x, y, \mathbf{f}^\star), x \neq null, y \neq null\cdot$$

$$P\_(x, w_1, \mathbf{f}) \vee P\_(x, w_2, \mathbf{f}) \vee P\_(x, w_3, \mathbf{f}) \;\leftarrow\; P\_(x, w_1, \mathbf{t}^\star), P\_(x, w_2, \mathbf{t}^\star), P\_(x, w_3, \mathbf{t}^\star), \bigwedge_{i \neq j} w_i \neq w_j, x \neq null, w_1 \neq null, w_2 \neq null, w_3 \neq null\cdot$$

We have a disjunctive solution program; actually one with two stable models, as expected according to Example 5.9. □

In (Barcelo et al. 2003; Bravo and Bertossi 2006), in the context of consistent query answering, several classes of ICs have been identified for which the logic program becomes *head-cycle free*, in which case, it can be replaced by and equivalent non-disjunctive, i.e. normal, program (Dantsin et al. 2001). Cautious reasoning from normal logic programs is *coNP*-complete (Dantsin et al. 2001). In consequence, for those classes of ICs, peer consistent query answering is in *coNP* in data complexity.[23] Among those constraints we find denial constraints, such as constraint (5) in Example 2.1. Even in the restricted import case with denial constraints as local constraints, peer consistent query answering is *coNP*-complete. (This can be obtained from CQA under denial constraints (Barcelo et al. 2003). Cf. (Bertossi 2011) for a survey of complexity results and references.)

7 Related Work

Our work can be placed among those on *semantic peer data management systems*, a research direction that was started, to the best of our knowledge, with (Halevy et al. 2003) (cf. also (Halevy et al. 2004)). Mappings relate two conjunctive queries that are expressed in terms of the schemas of two disjoint sets of peers. Already in those papers problematic cases of cyclic dependencies, which implicitly involve a cyclic accessibility graph, were identified. For example, we may have $\mathcal{P} = \{\mathtt{P1}, \mathtt{P2}, \mathtt{P3}\}$, 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, i.e. there is the implicit trust relation $\{(\mathtt{P1}, less, \mathtt{P2}), (\mathtt{P2}, less, \mathtt{P3}), (\mathtt{P3}, less, \mathtt{P1})\}$. This is an unrestricted and cyclic import case. In the presence of cycles like this, peer consistent query answering becomes undecidable.

In (Halevy et al. 2003), mappings are represented and given a semantics using classical first-order logic (FOL). This choice requires consistency of peers (i.e. their data, local ICs and local mappings) with respect to the system as a whole. More precisely, certain answers from a peer are defined as those true in every global instance that is consistent with the local data and metadata, which is rigid. Some

[23] Again, under the assumption that the peer has collected, for each of its neighbors, the intersection of its (local) solutions.

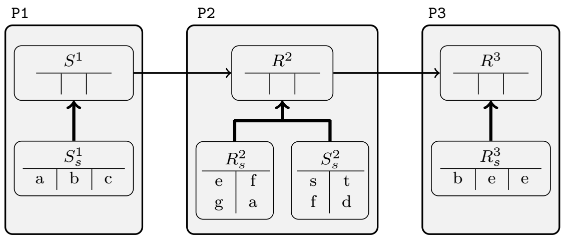


Fig. 8. Integration systems as peers

criticisms for the use of first-order logic as a basis for the semantics of PDESs were raised in (Calvanese et al. 2004), motivating a new research direction (cf. below).

For comparison, our approach is inconsistency tolerant, and does not provide a first-order semantics to the global system. Instead, we appeal to a non-monotonic semantics for only neighborhoods of peers. Our mappings are between two peers only, but in that case, more expressive than those considered in (Halevy et al. 2003).

It is worth noticing that all approaches adopting a FOL interpretation for PDESs, e.g. (Halevy et al. 2004; Halevy et al. 2003), do not consider peer mappings as constraints, but as logical implications from a peer to another. This in fact coincides with having trust relationships of the form $(\mathtt{P}, \mathit{less}, \mathtt{Q})$ for any mapping specified from P to Q.

Among the approaches to PDESs that are the closest to ours, we should mention (Calvanese et al. 2004; Calvanese et al. 2005; Calvanese et al. 2008; Franconi et al. 2004). The DECs are of the "import kind", and there are no explicit trust relationships. For example, in (Calvanese et al. 2008), DECs are of the form $CQ_i \rightarrow CQ_j$, where CQ_i, CQ_j are conjunctive queries over Pi and Pj's schemas, resp. This kind of DECs keep the schemas separate, on different sides of the implication. The latter has to be interpreted appropriately. Actually, the semantics of the system, in particular of the DECs, is given by an *epistemic logic*. Local ICs violations are avoided by ignoring a peer that is inconsistent with respect to 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. In consequence, a peer trust other peers more than itself, as long as no local inconsistencies are produced. There is not consistency restoration process involved.

As shown in Figure 8, in (Calvanese et al. 2008) each peer is considered as a local virtual data integration system that follows the *global-as-view* approach (Lenzerini 2002). In this example, the three peers are locally defined by the GAV mappings $\forall x \forall y \forall z(S^1_s(x,y,z) \rightarrow S^1(x,y,z))$, $\forall x \forall y \forall z(R^2_s(x,y) \wedge S^2_s(y,z) \rightarrow R^2(x,y,z))$, and $\forall x \forall y \forall z(R^3_s(x,y,z) \rightarrow R^3(x,y,z))$, resp. 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. In addition, the DECs establish mappings between the peers. In this case: $\forall x \forall y(R^2(x,y,z) \rightarrow \exists w S^1(x,y,w))$, $\forall x \forall y(R^3(x,y,y) \rightarrow \exists u \exists v R^2(u,x,v))$. These DECs are used to pass data from the peer mentioned in the antecedent to the one mentioned in the consequent. This activity is based on a theory written in epistemic logic which is generated from the local and inter-peer mappings. More specifically, the following epistemic theory is used for query answering:

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

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

$$\mathbf{K_3}(\forall x \forall y \forall z(R^3_s(x,y,z) \rightarrow R^3(x,y,z))) \quad \} \text{ Specification of } \texttt{P3}$$

$\mathbf{K_i}\varphi$ 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. (This semantics is similar in spirit to the one called *"send all"* in the discussion in the Online Appendix A.)

Our approach can be easily and uniformly adapted in order to make each peer a local data integration system. For this, the answer set programs that specify the legal instances of a virtual data integration introduced in (Bravo and Bertossi 2003; Bravo and Bertossi 2005; Bertossi and Bravo 2004a) can be used in combination with those presented here.

The epistemic theory has also been extended in (Calvanese et al. 2008) in order to make the PDES "inconsistency tolerant". This is done by using additional modal operators and extending the epistemic theory, achieving that: (a) A peer whose local data is inconsistent with respect to its ICs, is not considered for data exchange, that is, the PDES behaves as if that peer didn't exist. In our case, however, to such a peer, whenever possible, a repair semantics is applied. (b) Inconsistencies due to data imported from other peers, referred as P2P inconsistencies, are solved by removing a minimal amount of data imported from other peers in order to preserve consistency. In other words, atoms are imported into a peer by interaction with other peers only if this does not produce a violation of an integrity constraint of a peer. In our case, under the *same* or *less* trust conditions, the data is accepted and the peer applies again a local repair semantics.

In order to answer a query (Calvanese et al. 2005), 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 (Bravo 2007).

The semantics in (Franconi et al. 2004; Franconi et al. 2004) coincides with the epistemic semantics in (Calvanese et al. 2004). The former also 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.

The data exchange problem among distributed independent sources has been investigated in (Caroprese et al. 2006; Caroprese and Zumpano 2008; Caroprese and Zumpano 2011). In (Caroprese et al. 2006) the authors define a declarative semantics for P2P systems that allows one to import, into each peer, maximal subsets of atoms that do not violate the local integrity constraints. The framework has been extended in (Caroprese and Zumpano 2008) with a mechanism to set different degrees of reliability for neighbor peers, and in (Caroprese and Zumpano 2011), where "dynamic" preferences can be used to import data in the case of conflicting sets of atoms, depending on the properties of the peers' data. This extended framework allows one to model preferences like "in the case of conflicting information, it is preferable to import data from the neighbor peer that can provide the maximum number of tuples" or "in the case of conflicting information, it is preferable to import data from the neighbor peer such that the sum of the values of an attribute is minimum", without having to select preferred peers a-priori. To enforce this preference mechanisms the P2P framework has been enriched with aggregate functions.

In (Fuxman et al. 2006), the case of two peers that belong to a PDES is analyzed. From a source peer, P, to a target peer, Q, there may be both source-to-target dependencies, as in data exchange (Kolaitis 2005), and also target-to-source dependencies. The former are used to transport data from P to Q, and the second, to filter the received data in Q. In addition, the target may have an instance and local constraints. The existence of solutions for the target peer is investigated. The semantics we have introduced, but using labeled nulls instead of *null*, could be adapted to a situation like this (cf. the discussion in the Online Appendix A). It is also possible to use the solution programs in the case of (Fuxman et al. 2006). In this scenario, the source-to-target dependencies would give rise to program rules that do the data exchange, and both target-to-source dependencies and target dependencies would become hard program constraints.

More material and references on peer data exchange systems can be found in (Aberer 2011). Consistency issues that appear in our PDES scenario may also appear in the context of ontologies, when they have to be aligned in the presence of *bridges* between them. When ontologies are merged, it becomes necessary to solve inconsistencies between them (Serafini et al. 2005). However, in relation to mappings, we go beyond the cases considered in that area. The mappings, or bridges, between ontologies are in general much simpler than the general case considered in our work.

In this work we have concentrated on null values that are handled as in SQL relational databases. Actually, in that case, there is a single null, namely the constant

NULL. It can be interpreted in different ways in a database, e.g. as a representative for an existing but unknown value, as a missing or non-existent value, as a withheld or hidden value, etc. In our case, we do not make any ontological assumption about this constant. Our purpose consists in logically capturing the way it is handled in data management operations by an SQL database. More specifically, with respect to query answering, integrity checking and interaction with other data elements.

Most of the work in database theory bypasses the issue of SQL null, mainly because it is considered to be an undesirable oddity that could be replaced by a logically cleaner solution. However, SQL NULL exists, is used, and will most like stay with us. It would be better to accept it, but trying to clarify its logical and operational status. This is our approach in this work. From this point of view, we accept that SQL NULL appears, not only in databases, but could also appear explicitly in data exchange constraints and integrity constraints, and in their semantics. For example, a referential constraint or data exchange constraint could be of the form $\forall xy(S(x,y) \rightarrow T(x,\texttt{NULL}))$, i.e. requiring that T's second attribute should be filled with NULL.[24]

A similar approach is adopted in (Franconi and Tessaris 2012), where an alternative reconstruction of SQL NULL is attempted. They propose a corresponding relational algebra that takes care of the special status of NULL, but in the end, after the reconstruction, this constant disappears from the data domain. We keep it, but rewrite queries and constraints in such a way that it can be treated as any other constant.

There is a large literature on nulls in relational databases (see, e.g. (Levene and Loizou 1999)). However, to the best of our knowledge those nulls are "ideal" nulls, and not the real SQL NULL. There are also some logical approaches, starting with Reiter's treatment of nulls (Reiter 1984), which can be multiple, existential values not subject to the *unique names assumption*. They are also different from the SQL NULL. More recent work extending Reiter's approach can be found in (Traylor and Gelfond 1994; Lifschitz et al. 2012). See also (Libkin 2014; Libkin 2016) for a recent work on incomplete information in relational databases.

8 Conclusions

We have introduced a general framework for peer data exchange with trust relationships. The methodology is flexible and *inconsistency tolerant* in that each peer solves its data and semantic conflicts at query time, when querying its own and other peers' data.

The general semantic framework can be specialized in several ways, and we presented some possibilities. In particular, we developed a specific semantics based on universal and referential data exchange constraints, and on the use on SQL null values to deal with incomplete information. In this scenario, logic programs can

[24] In our case, given our repair semantics of Section 5, even a constraint of the form $\forall xy(S(x,y) \rightarrow \exists z T(x,z))$ will be enforced by giving to z the value NULL (unless some other constraint prevents this).

be used to specify the solution instances for a peer and to obtain peer consistent answers.

Techniques for optimizing the logic programs and their execution could be applied for query answering. Among them we find the partial evaluation of programs and the solution instances they specify, since we are not interested in the solution per se, but in the PCAs. More specifically, techniques used in CQA, such as *magic sets* for stable model semantics (Faber et al. 2007), and identification of predicates that are relevant to queries and constraints, could also be used in this setting. In this way, the number of rules and the amount of data that are needed to run the program are reduced (Caniupan and Bertossi 2010; Eiter et al. 2008).

The problem of query evaluation from disjunctive programs is Π_2^P-complete in data (Dantsin et al. 2001), which matches the complexity of PCA, as we established here. In spite of this, we have also identified syntactic classes of PDESs for which peer consistent query answering has a lower complexity. For them, specifically tailored mechanisms to solve this problem could be developed, as for CQA.

We should emphasize that the DECs we can handle are more general that those found in related work on peer data exchange.

Our semantics allows for inconsistent peers and inconsistencies between peers, without unraveling logical reasoning or having to exclude peers whose data participate in inconsistencies. Actually, our semantics for peer data exchange systems smoothly extend the repair semantics for consistent query answering from inconsistent databases.

Along the way, and interesting on its own merits, we developed a semantics of conjunctive query answering and constraint satisfaction in terms of classical FO predicate logic. It puts on a solid ground the mechanisms implemented in SQL databases and specifications in the SQL Standard for handling null values.

Acknowledgements: Research supported by NSERC DG (250279-2011) and a CITO/IBM-CAS Student Internship. Loreto Bravo is funded by CONICYT grant PSD-57. We appreciate valuable and detailed comments from the anonymous reviewers.

References

Aberer, K. 2011. *Peer-to-Peer Data Management.* Morgan & Claypool Publishers.

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases.* Addison-Wesley.

Arenas, M., Bertossi, L., and Chomicki, J. 1999. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM Symposium on Principles of Database Systems.* ACM, 68–79.

Arenas, M., Bertossi, L., and Chomicki, J. 2003. Answer Sets for Consistent Query Answers. *Theory and Practice of Logic Programming 3,* 4&5, 393–424.

Arieli, O., Denecker, M., and Bruynooghe, M. 2007. Distance Semantics for Database Repair. *Annals of Mathematics and Artificial Intelligence 50,* 3&4, 389–415.

Artz, D. and Gil, Y. 2007. A Survey of Trust in Computer Science and the Semantic Web. *Journal of Web Semantics 5,* 2, 58–71.

Barcelo, P. and Bertossi, L. 2003. Logic Programs for Querying Inconsistent Databases. In *Proc. Practical Aspects of Declarative Languages*. Springer, LNCS 2562, 208–222.

Barcelo, P., Bertossi, L., and Bravo, L. 2003. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In *Semantics of Databases*, Springer LNCS 2582, 7–33.

Bertossi, L. 2011. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers.

Bertossi, L. 2006. Consistent Query Answering in Databases. *ACM Sigmod Record 35,* 2 (June), 68–76.

Bertossi, L. and Bravo, L. 2004a. Consistent Query Answers in Virtual Data Integration Systems. In *Inconsistency Tolerance*. Springer, LNCS 3300, 42–83.

Bertossi, L. and Bravo, L. 2004b. Query Answering in Peer-to-Peer Data Exchange Systems. In *Proc. EDBT Workshop on Peer-to-Peer Computing and Databases*. Springer, LNCS 3268, 476–485.

Bertossi, L. and Bravo, L. 2007. The Semantics of Consistency and Trust in Peer Data Exchange Systems. In *Proc. International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*. Springer, LNCS 4790, 107–122.

Boutaba, R. and Marshall, A. (eds.) 2006. Special Issue on Management in Peer-to-Peer Systems. *Computer Networks 50,* 4, 472–484.

Bravo, L. 2007. Handling Inconsistency in Databases and Data Integration Systems. Ph.D. thesis, Carleton University, Department of Computer Science. http://people.scs.carleton.ca/~bertossi/papers/Thesis36.pdf

Bravo, L. and Bertossi, L. 2003. Logic Programs for Consistently Querying Data Sources. In *Proc. of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 10–15.

Bravo, L. and Bertossi, L. 2005. Disjunctive Deductive Databases for Computing Certain and Consistent Answers to Queries from Mediated Data Integration Systems. *Journal of Applied Logic 3,* 2, 329–367.

Bravo, L. and Bertossi, L. 2006. Semantically Correct Query Answers in the Presence of Null Values. In *Proc. EDBT WS on Inconsistency and Incompleteness in Databases*. Springer, LNCS 4254, 336–357.

Brewka, G., Eiter, T. and Truszczynski, M. 2011. Answer Set Programming at a Glance. *Commun. ACM 54,* 12, 92–103.

Cali, A., Lembo, D., and Rosati, R. 2003. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proc. ACM Symposium on Principles of Database Systems*. ACM Press, 260–271.

Calvanese, D., Damaggio, E., De Giacomo, G., Lenzerini, M., and Rosati, R. 2004. Semantic Data Integration in P2P Systems. In *Proc. VLDB International Workshop on Databases, Information Systems and Peer-to-Peer Computing*. Springer, LNCS 2944, 77–90.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. 2005. Inconsistency Tolerance in P2P Data Integration: An Epistemic Logic Approach. In *Proc. International Symposium on Database Programming Languages*. Springer, LNCS 3774, 90–105.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. 2008. Inconsistency Tolerance in P2P Data Integration: An Epistemic Logic Approach. *Information Systems 33,* 4-5, 360–384.

Calvanese, D., De Giacomo, G., Lenzerini, M., and Rosati, R. 2004. Logical Foundations of Peer-To-Peer Data Integration. In *Proc. ACM Symposium on Principles of Database Systems*. ACM Press, 241–251.

Caniupan, M. and Bertossi, L. 2010. The Consistency Extractor System: Answer Set Programs for Consistent Query Answering in Databases. *Data & Knowledge Engineering 69,* 6, 545–572.

Caroprese, L., Greco, S., and Zumpano, E. 2006. A Logic Programming Approach to Querying and Integrating P2P Deductive Databases. In *Proc. FLAIRS*. 31–36.

Caroprese, L. and Zumpano, E. 2008. Modeling Cooperation in P2P Data Management Systems. In *Proc. ISMIS*. 225–235.

Caroprese, L. and Zumpano, E. 2011. Aggregates and Priorities in P2P Data Management Systems. In *Proc. IDEAS*. 1–7.

Chomicki, J. 2007. Consistent Query Answering: Five Easy Pieces. In *Proc. International Conference on Database Theory*. Springer, LNCS 4353, 1–17.

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. 2001. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys 33,* 3, 374–425.

Demolombe, R. 2011. Transitivity and Propagation of Trust in Information Sources: An Analysis in Modal Logic. In *Proc. CLIMA XII*, Springer LNAI 6814, 13–28.

Eiter, T., Fink, M., Greco, G., and Lembo, D. 2008. Repair Localization for Query Answering from Inconsistent Databases. *ACM Transactions on Database Systems 33,* 2.

Eiter, T., Gottlob, G., and Mannila, H. 1997. Disjunctive Datalog. *ACM Transactions on Database Systems 22,* 3, 364–418.

Faber, W., Greco, G., and Leone, N. 2007. Magic Sets and their Application to Data Integration. *Journal of Computer and System Sciences 73,* 4, 584–609.

Franconi, E., Kuper, G., Lopatenko, A., and Serafini, L. 2004. A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems. In *Proc. VLDB Workshop on Databases, Information Systems and P2P Computing*. Springer, LNCS 2944, 64–76.

Franconi, E., Kuper, G., Lopatenko, A., and Zaihrayeu, I. 2004. A Distributed Algorithm for Robust Data Sharing and Updates in P2P Database Networks. In *Proc. EDBT Workshop on Peer-to-peer Computing and Databases*. Springer, LNCS 3268, 446–455.

Franconi, E. and Tessaris, S. 2012. On the Logic of SQL Nulls. In *Proc. Alberto Mendelzon WS on Foundations of Data Management*. Vol. 866. CEUR WS Proceedings.

Fuxman, A., Kolaitis, P., Miller, R., and Tan, W. 2006. Peer Data Exchange. *ACM Transactions on Database Systems 31,* 4, 1454–1498.

Gelfond, M. and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing 9*, 365–385.

Greco, G., Greco, S., and Zumpano, E. 2003. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Transactions on Knowledge and Data Engineering 15,* 6, 1389–1408.

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

Halevy, A., Ives, Z., Suciu, D., and Tatarinov, I. 2003. Schema Mediation in Peer Data Management Systems. In *Proc. International Conference on Data Engineering.* IEEE Computer Society, 505–518.

Imielinski, Th., and Lipski, W. 1984. Incomplete Information in Relational Databases. *Journal of the ACM 31,* 4: 761–791.

Jøsang, A., Marsh, S., and Pope, S. 2006. Exploring Different Types of Trust Propagation. In *Proc. iTrust*. Springer, LNCS 3986, 179–192.

Jøsang, A., Azderska, T. and Marsh, S. 2012. Trust Transitivity and Conditional Belief Reasoning. In *Proc. IFIPTM 2012, IFIP AICT 374*. Springer, 68–83.

Kolaitis, P. 2005. Schema Mappings, Data Exchange, and Metadata Management. In *Proc. ACM Symposium on Principles of Database Systems.* ACM Press, 61–75.

Lenzerini, M. 2002. Data Integration: A Theoretica Perspective. In *Proc. ACM Symposium on Principles of Database Systems.* ACM Press, 233–246.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562.

Levene, M. and Loizou, G. 1997. Null Inclusion Dependencies in Relational Databases. *Information and Computation 136,* 2, 67–108.

Levene, M. and Loizou, G. 1999. *A Guided Tour of Relational Databases and Beyond.* Springer.

Libkin, L. 2014. Incomplete Data: What Went Wrong, and How to Fix It. In *Proc. ACM Symposium on Principles of Database Systems.* ACM Press, 1–13.

Libkin, L. 2016. Certain Answers as Objects and Knowledge. *Artificial Intelligence 232,* 1, 1–19.

Lifschitz, V., Pichotta, K., and Yang, F. 2012. Relational Theories with Null Values and Non-Herbrand Stable Models. *Theory and Practice of Logic Programming*, 12(4-5):565-582.

Marti, S. and Garcia-Molina, H. 2006. Taxonomy of Trust: Categorizing P2P Reputation Systems. *Computer Networks 50,* 4, 472–484.

Papadimitriou, Ch. 1994. *Computational Complexity.* Addison-Wesley.

Reiter, R. 1984. Towards a Logical Reconstruction of Relational Database Theory. In *On Conceptual Modelling*, M.Brodie, J.Mylopoulos, and J. Schmidt, Eds. Springer, 191–233.

Sabater, J. and Sierra, C. 2005. Review on Computational Trust and Reputation Models. *Artificial Intelligence Review 24,* 1, 33–60.

Serafini, L., Borgida, A., and Tamilin, A. 2005. Aspects of Distributed and Modular Ontology Reasoning. In *Proc. International Joint Conference on Artificial Intelligence.* Morgan Kaufmann, 570–575.

Traylor, B. and Gelfond, M. 1994. Representing Null Values in Logic Programming. In *Logical Foundations of Computer Science.* Springer LNCS 813, 341–352.

ONLINE APPENDIX for the paper *Consistency and Trust in Peer Data Exchange Systems*

published in Theory and Practice of Logic Programming

LEOPOLDO BERTOSSI
Carleton University, School of Computer Science, Ottawa, Canada.
bertossi@scs.carleton.ca

LORETO BRAVO
Universidad del Desarrollo, Facultad de Ingeniería, Santiago, Chile.
bravo@udd.cl

Appendix A Discussion

A.1 On cycles and their assumptions

In this section, unless stated otherwise, we refer to the special semantics introduced in Section 4.

We have assumed that $\mathcal{G}(\mathfrak{P})$ is acyclic. However, the peers, not being aware of being in a cycle in $\mathcal{G}(\mathfrak{P})$, could attempt to do data exchange as described above. In order to detect an infinite loop, for a query $\mathcal{Q}$ posed by a peer $\mathtt{P}$, a unique identifier $id(\mathtt{P}, \mathcal{Q})$ can be created and kept in all the queries that have origin in $\mathcal{Q}$. If an identifier comes back to a peer, it will realize that it is in a cycle and act accordingly.

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

Example 1
Consider $\mathcal{S}(\mathtt{P1}) = \{R^1(\cdot), S^1(\cdot)\}$, $\mathcal{S}(\mathtt{P2}) = \{R^2(\cdot),\ S^2(\cdot)\}$, $\Sigma(\mathtt{P1},\ \mathtt{P2}) = \{\forall x(R^2(x) \rightarrow R^1(x))\}$, $\Sigma(\mathtt{P2}, \mathtt{P1}) = \{\forall x(S^1(x) \rightarrow S^2(x))\}$, $\mathit{Trust} = \{(\mathtt{P1},\ \mathit{less},\ \mathtt{P2}),\ (\mathtt{P2},\ \mathit{less},\ \mathtt{P1})\}$. If a query is posed to $\mathtt{P1}$, it will request from $\mathtt{P2}$ the PCAs to query $R^2(x)$, but not those to query $S^2(x)$. Peer $\mathtt{P2}$ can realize it does not need data from $\mathtt{P1}$ and will simply return $D(\mathtt{P2})\!\downharpoonright\!\{R^2\}$ to $\mathtt{P1}$, 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 in the query answering process. $\Box$

As we mentioned before, if there are ref-cycles in $\Sigma(\texttt{P})$, the stable models of the solution program for `P` may correspond to a strict superset of the solutions. This is shown in the next example. In such a case, post-processing that deletes models corresponding to non-minimal "solutions" is necessary.

Example 2
Consider $D(\texttt{P1}) = \{R^1(a,b)\}$, $D(\texttt{P2}) = \{R^2(a,c)\}$, $\Sigma(\texttt{P1},\texttt{P2}) = \{\forall x \forall z(R^1(x,z) \rightarrow \exists y\, R^2(x,y)), \forall x \forall z(R^2(x,z) \rightarrow \exists y\, R^1(x,y)\}$, which is ref-cyclic; $\Sigma(\texttt{P2}) = \Sigma(\texttt{P1},\texttt{P1}) = \emptyset$; and $(\texttt{P1}, \mathit{same}, \texttt{P2}) \in \mathit{Trust}$. Here, `P1` has only one solution, namely $\{R^1(a,b)\}$.

However, $\Pi(\texttt{P1})$ has two models. The most relevant part of the program consists of the facts $R^1(a,b)$, $R^2(a,c)$, and the following rules:

$$\begin{array}{rcl}
R^1\_(x,y,\mathbf{f}) \vee R^2\_(x,\mathit{null},\mathbf{t}) & \leftarrow & R^1\_(x,y,\mathbf{t}^\star), \mathit{not}\ \mathit{aux}_1(x), x \neq \mathit{null} \\
\mathit{aux}_1(x) & \leftarrow & R^2(x,\mathit{null}), \mathit{not}\, R^2\_(x,\mathit{null},\mathbf{f}) \\
\mathit{aux}_1(x) & \leftarrow & R^2(x,y,\mathbf{t}^\star), \mathit{not}\, R^2\_(x,y,\mathbf{f}), x \neq \mathit{null}, y \neq \mathit{null} \\
R^2\_(x,y,\mathbf{f}) \vee R^1\_(x,\mathit{null},\mathbf{t}) & \leftarrow & R^2\_(x,y,\mathbf{t}^\star), \mathit{not}\ \mathit{aux}_2(x), x \neq \mathit{null} \\
\mathit{aux}_2(x) & \leftarrow & R^1(x,\mathit{null}), \mathit{not}\, R^1\_(x,\mathit{null},\mathbf{f}) \\
\mathit{aux}_2(x) & \leftarrow & R^1(x,y,\mathbf{t}^\star), \mathit{not}\, R^1\_(x,y,\mathbf{f}), x \neq \mathit{null}, y \neq \mathit{null}
\end{array}$$

The two models correspond to the neighborhood "solutions" $\{R^1(a,b), R^2(a,c)\}$ and $\emptyset$, producing in their turn, the instances $\{R^1(a,b)\}$ and $\emptyset$, resp., for `P1`. Only the former is 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 $R^2\_(a,c,\mathbf{f})$ to be true, then by the second and third rules above, $\mathit{aux}_1(a)$ is false. (ii) Then, the first rule can be satisfied, by making $R^1\_(a,b,\mathbf{f})$ true. (iii) By the fifth and sixth rules, $\mathit{aux}_2(a)$ is false. (iv) This justifies making $R^2\_(a,b,\mathbf{f})$ true, thus, closing the cycle. Notice, that in the whole justification the changes where not determined by inconsistencies. $\Box$

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.

Example 3
(example 2 continued) If we replace $(\texttt{P1}, \mathit{same}, \texttt{P2}) \in \mathit{Trust}$ by $(\texttt{P1}, \mathit{less}, \texttt{P2}) \in \mathit{Trust}$, the relevant part of $\Pi(\texttt{P1})$ now is: $R^1(a,b)$, $R^2(a,c)$, plus

$$\begin{array}{rcl}
R^1\_(x,y,\mathbf{f}) & \leftarrow & R^1\_(x,y,\mathbf{t}^\star), \mathit{not}\ \mathit{aux}_1(x), x \neq \mathit{null}\, \cdot \\
\mathit{aux}_1(x) & \leftarrow & R^2(x,\mathit{null}), \mathit{not}\, R^2\_(x,\mathit{null},\mathbf{f})\, \cdot \\
\mathit{aux}_1(x) & \leftarrow & R^2(x,y,\mathbf{t}^\star), \mathit{not}\, R^2\_(x,y,\mathbf{f}), x \neq \mathit{null}, y \neq \mathit{null}\, \cdot \\
R^1\_(x,\mathit{null},\mathbf{t}) & \leftarrow & R^2\_(x,y,\mathbf{t}^\star), \mathit{not}\ \mathit{aux}_2(x), x \neq \mathit{null}\, \cdot \\
\mathit{aux}_2(x) & \leftarrow & R^1(x,\mathit{null}), \mathit{not}\, R^1\_(x,\mathit{null},\mathbf{f})\, \cdot \\
\mathit{aux}_2(x) & \leftarrow & R^1(x,y,\mathbf{t}^\star), \mathit{not}\, R^1\_(x,y,\mathbf{f}), x \neq \mathit{null}, y \neq \mathit{null} \cdot
\end{array}$$

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. □

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, for cyclic accessibility graphs, *super peers* (Yang and Garcia-Molina 2003) could be used, to detect cycles and prune certain DECs, making the graph acyclic if necessary; and then our semantics could be applied.

A.2 Query sensitive query answering

Our definition of the solution semantics and the peer consistent answers might suggest that, in order to answer a particular query $\mathcal{Q}$, a peer P has to import the full intersection of the solutions of each of its neighbors, which in their turn have to do the same, etc. If we do this, most likely most of the data imported by P will be irrelevant for the query at hand, and is not needed.

It is possible to design query answering methodologies that are more sensitive to the query at hand, in the sense that only the relevant data is transitively imported into P before answering $\mathcal{Q}$. A full treatment of this subject is beyond the scope of this paper. However, we can give some indications as to how to proceed.

In (Caniupan and Bertossi 2010), the *dependency graph* of database predicates with respect to a set of ICs was introduced and used to capture the notion of possibly transitive relevance of a predicate for another, which is useful in consistent query answering. Here we can use similar graphs for each peer in relationship with its neighbors through a set of DECs, also taking into account the local ICs. These graphs would give us a better upper bound on what to import from other peers (as opposed to bringing the full intersection of solutions).

More precisely, if a query $\mathcal{Q}$ to P contains $\mathcal{S}(\mathtt{P})$-predicates $P_1,\ldots,P_n$, with relevant $\mathcal{S}(\mathtt{Q})$-predicates $Q_1^1,\ldots,Q_{m_1}^1,\ldots,Q_1^n,\ldots,Q_{m_n}^n$, resp., at a neighbor Q, then P will request from Q the PCAs to each of the constant-free, atomic queries $Q_j^i(\bar{x})$. The corresponding sets of answers will form the (most likely smaller) instance provided by Q to P, who will prune its repair program by keeping only the relevant rules, i.e. those that are related to $\mathcal{Q}$, the P_i and the Q_j^i. This idea can be illustrated by means of an example.

Example 4
Consider a schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$ with $\mathcal{P} = \{\mathtt{P1},\mathtt{P2},\mathtt{P3}\}$, $\mathfrak{S} = \{\mathcal{S}(\mathtt{P1}), \mathcal{S}(\mathtt{P2}), \mathcal{S}(\mathtt{P3})\}$, $\mathcal{S}(\mathtt{P1}) = \{R^1(\cdot,\cdot),\ S^1(\cdot,\cdot),\ T^1(\cdot,\cdot)\}$, $\mathcal{S}(\mathtt{P2}) = \{R^2(\cdot,\cdot),\ S^2(\cdot,\cdot), T^2(\cdot,\cdot)\}$, $\mathcal{S}(\mathtt{P3}) = \{R^3(\cdot,\cdot), S^3(\cdot,\cdot)\}$, $\mathit{Trust} = \{(\mathtt{P1}, \mathit{less}, \mathtt{P2}),\ (\mathtt{P2}, \mathit{same}, \mathtt{P3})\}$. Let $\mathfrak{D}$ be an arbitrary instance for the PDES.

The sets of DECs are: $\Sigma(\mathtt{P1},\ \mathtt{P1}) = \{\forall x \forall y(R^1(x,y) \land S^1(x,y) \rightarrow \mathbf{false})\}$, $\Sigma(\mathtt{P1},\mathtt{P2}) = \{\forall x \forall y(R^2(x,y) \rightarrow R^1(x,y)),\ \forall x \forall y(S^2(x,y) \rightarrow S^1(x,y))\}$, and $\Sigma(\mathtt{P2}, \mathtt{P3}) = \{\forall x \forall y(R^2(x,y) \land R^3(x,y) \rightarrow \mathbf{false})\}$.

If P1 is posed the query $\mathcal{Q}_1(x) :\ \exists y R^1(x,y)$, then the relevant predicates in

$\mathcal{S}(\texttt{P1})$ are R^1, S^1 (due to the DEC in $\Sigma(\texttt{P1})$). Then, through the DECs also, it follows that the predicates that are relevant to P1 are R^2, S^2 at P2. So, P1 poses to P2 the queries $R^2(x,y), S^2(x,y)$. The only relevant predicate at P3 is S^3. So, P2 will pose to P3 the query $S^3(x,y)$.

In this case, P3 will return $D(\texttt{P3}) \downharpoonright \{R^3\}$ to P2, which, due to the UDEC in $\Sigma(\texttt{P2},\texttt{P3})$, will subtract it from $D(\texttt{P2}) \downharpoonright \{R^2\}$, because $(\bigcap Sol(\texttt{P2}, \mathfrak{D})) \downharpoonright \{R^2\} = (D(\texttt{P2}) \downharpoonright \{R^2\} \smallsetminus D(\texttt{P3}) \downharpoonright \{R^3\})$. Peer P2 will send this difference to P1 as it is the answer to query $R^2(x,y)$. Peer P2 will also return to P1 the entire $D(\texttt{P2}) \downharpoonright \{S^2\}$ as its answer to the query $S^2(x,y)$.

Finally, P1 will answer the original query with a solution program containing as facts the tuples in $D(\texttt{P1}) \downharpoonright \{R^1\}, D(\texttt{P1}) \downharpoonright \{S^1\}, D(\texttt{P2}) \downharpoonright \{S^2\}, ((D(\texttt{P2}) \downharpoonright \{R^2\}) \smallsetminus (D(\texttt{P3}) \downharpoonright \{R^3\}))$. The last set, as an extension for R^2 in the program. □

The methodology sketched in this example will be certainly more efficient than computing and shipping the full intersection of a peer's solutions. It is natural to expect that additional optimizations can be developed.

A particularly appealing, but provably less general, approach to peer-consistent query answering is *first-order query rewriting*, which we illustrate by means of an example.

Example 5
Consider an instance $\mathfrak{D} = \{D(\texttt{P1}), D(\texttt{P2}), D(\texttt{P3})\}$ for the schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$ with $\mathcal{P} = \{\texttt{P1},\texttt{P2},\texttt{P3}\}$, $\mathfrak{S} = \{\mathcal{S}(\texttt{P1}), \mathcal{S}(\texttt{P2}), \mathcal{S}(\texttt{P3})\}$, $\mathcal{S}(\texttt{Pi}) = \{R^i(\cdot,\cdot)\}$, $\mathit{Trust} = \{(\texttt{P1}, \mathit{less}, \texttt{P2}), (\texttt{P1}, \mathit{same}, \texttt{P3})\}$, $D(\texttt{P1}) = \{R^1(a,b),\ R^1(s,\ t)\}$, $D(\texttt{P2}) = \{R^2(c,d),\ R^2(a,e)\}$, $D(\texttt{P3}) = \{R^3(a,f),\ R^3(s,u)\}$ and the DECs:

$$\begin{aligned}\Sigma(\texttt{P1},\texttt{P2}) &= \{\forall x \forall y (R^2(x,y) \rightarrow R^1(x,y))\};\\ \Sigma(\texttt{P1},\texttt{P3}) &= \{\forall x \forall y \forall z (R^1(x,y) \wedge R^3(x,z) \rightarrow y = z)\}.\end{aligned}$$

We are interested in P1's solutions. Since P2, P3 are sink peers in the graph $\mathcal{G}(\mathfrak{P})$, we have the extended instance $D = \{R^1(a,b), R^1(s,t), R^2(c,d), R^2(a,e), R^3(a,f), R^3(s,u)\}$, from which we have to obtain the solutions for P1.

The solutions for P1 are obtained by first repairing D with respect to $\Sigma(\texttt{P1},\texttt{P2})$, obtaining $D_1 = \{R^1(a,b), R^1(s,t),\ R^1(c,d), R^1(a,e),\ R^2(c,d), R^2(a,e),\ R^3(a,f), R^3(s,u)\}$. We have only one repair at this stage, which now has to be repaired in its turn with respect to $\Sigma(\texttt{P1},\texttt{P3})$ (but keeping the relationship between P1 and P2 satisfied). There are two sets of tuples violating $\Sigma(\texttt{P1},\texttt{P3})$ in D_1: $\{R^1(s,t), R^3(s,u)\}$ and $\{R^1(a,b), R^1(a,e),\ R^3(a,f)\}$. The first violation can be repaired by deleting any, but only one, of the two tuples. The second one, by deleting tuple $R^3(a,f)$ only (otherwise we would violate the relationship between P1 and P2).

As a consequence, we obtain two neighborhood solutions: $D' = \{R^1(a,b), R^1(s,t), R^1(c,d),\ R^1(a,e), R^2(c,d), R^2(a,e)\}$, and $D'' = \{R^1(a,b), R^1(c,d), R^1(a,e), R^2(c,d), R^2(a,\ e),\ R^3(s,u)\}$. The solutions for P1 are: $D(\texttt{P1})' = \{R^1(a,b), R^1(s,t), R^1(c,d),\ R^1(a,e)\}$ and $D(\texttt{P1})'' = \{R^1(a,b),\ R^1(c,d),\ R^1(a,e)\}$.

If the query $\mathcal{Q} : R^1(x,y)$ is posed to P1, the PCAs are $\langle a,b\rangle, \langle c,d\rangle, \langle a,e\rangle$, because those are R^1-tuples found in in the intersection of P1's solutions.

Now, let us try an alternative method for peer consistently answering the same query. We first rewrite the query using the DEC in $\Sigma(\texttt{P1},\texttt{P2})$, obtaining $\mathcal{Q}'$: $R^1(x,y) \vee R^2(x,y)$, with the effect of bringing P2's data into P1. Next, considering $\Sigma(\texttt{P1},\texttt{P3})$, this query is rewritten as

$$\mathcal{Q}'' : [R^1(x,y) \wedge \forall z_1((R^3(x,z_1) \wedge \neg\exists z_2 R^2(x,z_2)) \rightarrow z_1 = y)] \vee R^2(x,y)\cdot \quad \text{(A1)}$$

To answer this query, P1 first issues a query to P2 to retrieve the tuples in R^2, since they will be essentially in R^1 in all the solutions, due to $\Sigma(\texttt{P1},\texttt{P2})$. Next, a query is issued to P3 to leave aside from the answers those tuples of R^1 that have the same first but not the same second argument in R^3. This filtering is performed as long as there is no tuple in R^2 that "protects" the tuple in R^1. For example, the tuple $R^1(a,b)$ is protected by $R^2(a,e)$ because, as $R^1(a,e)$ belongs to all the solutions, the only way to repair a violation with respect to $\Sigma(\texttt{P1},\texttt{P3})$ is by deleting the tuple from R^3. In this case, the R^1-tuple will be in the answer.

We can see that answering query (A1) amounts to issuing from P1 queries to P2, P3 about the contents of their relations, R^2 and R^3, resp., which are answered by the latter by classical query evaluation over their local instances. After those data have been gathered by P1, it proceeds to evaluate (A1), which contains an implicit repair process.

Now, the answers to (A1) are $\langle a,b\rangle, \langle c,d\rangle, \langle a,e\rangle$, precisely the PCAs we obtained above, considering all the explicit solutions for P1. □

The FO rewriting methodology we just illustrated is bound to have limited applicability. If this was a general mechanism, PCAs to conjunctive queries could be obtained in polynomial time in data, i.e. in the size of the union of the instances of a peer and those of its neighbors (or the local intersections of their solutions). However, Corollary 5.2 tells us that the complexity is higher than this (if $P \neq NP$).

A.3 A semantics based on arbitrary data elements

The purpose of this section is twofold. First, we will present an alternative special semantics that fits into the general semantic framework of Section 3. Second, we will show that this general semantics (and also the one in Section 4) can handle data mappings that are more complex that those usually considered in the related work on peer data exchange. All this will be done on the basis of an extended example.

Consider a PDES $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$ with $\mathcal{P} = \{\texttt{P1},\texttt{P2}\}$, $\mathfrak{S} = \{\mathcal{S}(\texttt{P1}), \mathcal{S}(\texttt{P2})\}$, $\mathcal{S}(\texttt{P1}) = \{R^1(\cdot,\cdot),\ T^1(\cdot,\cdot)\}$, $\mathcal{S}(\texttt{P2}) = \{T^2(\cdot,\cdot),\ S^2(\cdot,\cdot)\}$, $\mathit{Trust} = \{(\texttt{P1}, \mathit{less}, \texttt{P2})\}$, and $\Sigma(\texttt{P1},\texttt{P2})$ consists of the following DEC:

$$\forall x \forall y \forall z (R^1(x,y) \wedge T^2(z,y) \rightarrow \exists w (T^1(x,w) \wedge S^2(z,w)))\cdot \quad \text{(A2)}$$

This DEC, which falls within our general syntactic class of DECs, mixes tables of the two peers on each side of the implication. This kind of mapping is more general than those typically considered in virtual data integration (Lenzerini 2002) or data exchange (Kolaitis 2005).[25]

[25] Cf. (Bertossi and Bravo 2004b) for some connections between PDESs and virtual data integration under the *local-as-view* approach. Also (De Giacomo et al. 2007), for relationships between PDESs, virtual data integration, and data exchange.

If (A2) is not satisfied by the data in P1 and P2, which happens when the join in the antecedent is satisfied, but not the one in the consequent, then solutions for P1 have to be found, keeping P2's data fixed in the process, due to the trust relationship. Now, in this section we will depart from the solution semantics introduced in Section 4, by restoring consistency with respect to (A2) through the introduction of arbitrary elements of the data domain $\mathcal{U}$.[26] Those elements become witnesses for the existentially quantified variables in the DECs. That is, these values come from the data domain, and are not replaced by *null* or by labeled nulls as in data exchange (Kolaitis 2005). Since we have alternative choices for them, we may obtain different solutions for a peer. However, by definition of solution, they have to stay as close as possible to the original instance. In this case, the general comparison relation $\preceq_D$ between neighborhood instances of Section 3 is given by: $D_1 \preceq_D^\Delta D_2$ iff $\Delta(D, D_1) \subseteq \Delta(D, D_2)$.

We will specify the solutions for this example directly in (or using) logic programs. We will also show the most relevant part of the program $\Pi^-(\texttt{P1})$. Since we have to restore consistency with respect to (A2), the main rules are (A3)-(A6) below.

$$R^1\_(x,y,\mathbf{f}) \leftarrow R^1\_(x,y,\mathbf{t}^\star),\, T^2(z,y),\, \mathit{not}\ \mathit{aux}_1(x,z), \mathit{not}\ \mathit{aux}_2(z) \cdot \tag{A3}$$

$$\mathit{aux}_1(x,z) \leftarrow T^1\_(x,w,\mathbf{t}^\star), S^2(z,w) \cdot \tag{A4}$$

$$\mathit{aux}_2(z) \leftarrow S^2(z,w) \cdot \tag{A5}$$

That is, $R^1(x,y)$ is deleted if it participates in a violation of (A2) (what is captured by the first three literals in the body of (A3) plus rule (A4)), and there is no way to restore consistency by inserting a tuple into T^1, because there is no possible matching tuple in S^2 for the possibly new tuple in T^1 (what is captured by the last literal in the body of (A3) plus rule (A5)). In case there is such a tuple in S^2, we can either delete a tuple from R^1 or insert a tuple into T^1:

$$\begin{aligned} R^1\_(x,y,\mathbf{f}) \vee T^1\_(x,w,\mathbf{t}) \leftarrow\ & R^1\_(x,y,\mathbf{t}^\star),\ T^2(z,y),\ \mathit{not}\ \mathit{aux}_1(x,z), \\ & S^2(z,w),\ \mathit{choice}((x,z),w) \cdot \end{aligned} \tag{A6}$$

That is, in case of a violation of (A2), when there is tuple of the form $S^2(a,t)$ in S^2 for the combination of values $\langle d,a \rangle$, then the *choice operator* (Giannotti et al. 1991) non-deterministically chooses a unique value for t, so that the tuple $T^1(d,t)$ is inserted into T^1 as an alternative to deleting $R^1(d,m)$ from R^1. The *choice* predicate can be eliminated and replaced by another predicate that can be specified by means of extra but standard rules (Giannotti et al. 1991).

If, instead, we had $\mathit{Trust} = \{(\texttt{P1}, \mathit{same}, \texttt{P2})\}$, P2's relations would also be flexible when searching for solution instances. In this case, the program becomes more involved in terms of presentation (but not conceptually), in the sense that more relations can be updated, and corresponding repair rules have to be added.

Notice that in this example, the values that are chosen as witnesses for the existential quantifier in the DEC are taken from the active domain of the database,

[26] This $\mathcal{U}$ could be a finite superset of the union of the active domains involved or infinite. The latter case is also covered by our semantics. The logic programming semantics is also perfectly defined in this case.

namely from the set of values for the second attribute of relation S^2. In other cases, for example with a DEC of the form $\forall x \forall y(T^2(x, y) \rightarrow \exists z R^1(x, z))$, we have to consider arbitrary values from an underlying domain *dom* when inserting tuples into R^1. In this case, *dom* requires a specification as a finite predicate in the program.

Some of the ideas presented above (such as the insertion of elements from the active domain and the use of the choice operator) have been fully developed and applied by the authors (Bertossi and Bravo 2004a; Bravo and Bertossi 2003; Bravo and Bertossi 2005) to inconsistency management in virtual data integration systems under the *local-as-view approach* (Lenzerini 2002).

A.4 Data transport and semantics

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. In the context of the general semantic framework introduced in Section 3, we developed a particular semantics in Section 4, according to which a peer `Q` passes back to a neighbor `P`, who is building its solutions, this is (part of) its certain data. This is the one that holds in all of `Q`'s solutions.

In (Bravo 2007, chapter 7) also two other alternative semantics are fully developed and compared, in particular establishing some conditions under which they coincide or differ. Those other semantics assume that more detailed information, such as mappings and trust relationships, can be sent between peers. We briefly describe them.

1. Send all. The first one assumes that data, DECs and trust relations can be sent between peers. So, we can think that we have a possibly large database instance that has to be virtually repaired in order to satisfy all the relevant DECs (obtained from the accessibility graph) and at the same time accommodating the trust relationships. In this case, the DECs are treated as traditional ICs on the integrated instance. This semantics is similar to the repair semantics for consistent query answering. Preferences imposed on repairs can be used to capture the trust relationships. In this case, it is not necessary to require the acyclicity of the accessibility graph.

2. Send solutions. The second one assumes that only solutions can be send between neighboring peers. In this case, a peer `P` requests the solutions of the neighbors in order to calculate its own solutions. Here, the database consists of the data at `P` plus all the solutions of `P`'s neighbors; and the constraints are the DECs between `P` and its neighbors. As in Section 3, this is a recursive definition, and assumes an acyclic accessibility graph.

We think that the semantics we developed in Section 3, which could be called *"send cores"*, is more natural (a peer passes over what it is certain about), and also simplifies reasoning at the local level, i.e. at each peer's site, because at most one instance peer neighbor has to be considered.

Now, under our official semantics, if we want to use local solution programs, each neighbor of a peer `P` has to run its program, and then send the (relevant part of the) intersection of the stable models to `P`, who runs its local solution program. This means that different programs have to be fed externally.

At least under the *"send solutions"* and *"send cores"* semantics (which assumes an acyclic peer graph), we could imagine having a single program that does all this output/input concatenation, *internally*. Actually, it is possible to build a *single program for a peer*, say $\Pi^{man}(\texttt{P})$, that acts as a combination of solution programs as given in Section 6. For each peer `Q` that is accessible from `P`, the program locally runs a solution program, produces `Q`'s solutions (its stable models), or the intersection thereof; and, without leaving $\Pi^{man}(\texttt{P})$, passes them to its preceding neighbors `Q'`, who uses them to locally compute its own solutions by means of its own, local solution program, etc., until reaching `P`. Such a program $\Pi^{man}(\texttt{P})$ can be a *manifold program* (Faber and Woltran 2011).

Actually, within a manifold program (MP), a program can pass certain or possible query answers as an input to another program. Conceptually, MPs offer a nice logical solution, by means of a single program, to this form of program combination that, otherwise, would require external intervention (the efficient implementation of MPs is still an open problem).

A.5 On trust

We introduced trust relationships in the process of peer data exchange already in (Bertossi and Bravo 2004b); and here we have further developed this idea, in a general semantic framework. However, the notion of trust we have in this work is still rather simple. Actually, it can be represented as an annotated binary relation between peers. It would be interesting to impose a more sophisticated and rich model of trust on top of the DECs-bases network of peers. Our concern is not about computing or updating trust in a P2P overlay network (Xiong and Liu 2004; Jøsang et al. 2006), but about logical specifications of trust. We envision a logic-based model of trust that can be integrated with/into the DECs. Such a model could express some higher level properties, e.g. symmetry or transitivity of trust relationships. A logic-based representation of trust could allow us to infer non-explicit trust relationships whenever necessary.

Trust modeling is an active and important area of research nowadays, most notably in the context of the semantic web. See (Sabater and Sierra 2005; Artz and Gil 2007) for surveys. The integration of trust models, including related notions, like reputation, provenance, etc., into peer data exchange is still an open area of research. This is specially the case for logic-based models of trust (Herzig et al. 2008). See (Hien Nguyen et al. 2008) and references therein for probabilistic approaches.

In Definition 6.1, the trust relationships between peers were implicitly and rigidly captured in the specifications of solutions by means of the disjunctive heads of the repair rules. Although this is a simpler way of presenting things, it has some drawbacks: (a) The approach is not modular in the sense that trust is built-in into the rules; (b) Changes in trust relationships requires changing heads of repair rules;

and (c) In case no solution exists due to the rigid and conflicting trust relationships, no alternative, but possibly less desirable solution can be obtained as a stable model of the solution program.

One way of addressing these issues is through the use of *preference programs*, which are answer set programs that express different forms of preference, which essentially amounts to preferring and keeping only certain stable models of the program. For example, in a disjunctive rule of the form $A \vee B \leftarrow Body$, one could prefer to make A true instead of B. If this is possible, only that stable model would be chosen. However, if that is not possible (due to the other rules and facts in the program), making B true is still good enough. More complex preferences could also be captured. Preferences can be explicitly and declaratively expressed, and the resulting programs can be compiled into usual answer set programs with their usual stable model semantics (cf. (Brewka 2004) and references therein).

Here we briefly outline how *weak program constraints* (Buccafurri et al. 2000; Leone et al. 2006) declaratively capture the kind of preferences that address our needs. (Cf. (Brewka 2004) for connections between preferences in logic programs and weak constraints.).

Example 6
Consider Example 6.2, where $(\mathtt{P1}, less, \mathtt{P2}) \in Trust$ is captured by the non-disjunctive repair rule

$$R^1\_(x, y, \mathbf{t}) \leftarrow R^2\_(x, y, \mathbf{t}^\star), R^1\_(x, y, \mathbf{f}^\star), x \neq null, y \neq null\cdot$$

The same effect, and more, could be obtained by uniformly using disjunctive rules followed by appropriate weak constraints. In this case,

$$\begin{array}{rcl} R^1\_(x, y, \mathbf{t}) \vee R^2\_(x, y, \mathbf{f}) & \leftarrow & R^2\_(x, y, \mathbf{t}^\star), R^1\_(x, y, \mathbf{f}^\star), x \neq null, y \neq null\ \cdot \\ & \Leftarrow & R^2\_(x, y, \mathbf{f})\cdot \end{array} \qquad \text{(A7)}$$

Here, the weak constraint (A7) expresses a preference for the stable models of the program that minimize the number of violations of the condition expressed its body, in this case, that, when restoring the satisfaction of the DEC $\forall x \forall y (R^2(x, y) \rightarrow R^1(x, y))$, the tuple $R^2(x, y)$ is not deleted. These weak constraints are used by a peer `P` to ensure that, if possible, the tuples in the peers that it trusts more than itself are not modified. □

If the original solution program has solutions, then the new program would have the same solutions. However, the latter could have solutions when the former does not. This would make the semantics of the system more flexible with respect to unsatisfiable trust requirements. It is also clear that the weak constraints could be easily derived from the trust relationships and the DECs. The solution program with weak constraints can be run in the *DLV* system (Leone et al. 2006) to obtain the solutions and peer consistent answers of a peer.

Notice that the new repair programs, except for the weak program constraints, are now of the same kind as those for specifying repairs of single databases with respect to local ICs (Bravo and Bertossi 2006). Actually, if in the new program the weak program constraints are replaced by (hard) program constraints, e.g. (A7) by $\leftarrow R^2\_(x, y, \mathbf{f})$, the solutions coincide with those of the programs in Definition 6.1.

We should mention that in (Arenas et al. 2003), weak constraints were used, as a part of a repair program, to specify the preference for *cardinality repairs*, i.e. repairs that minimize the *number* of tuples that are inserted or deleted to restore consistency, as opposed to minimality (with respect to subset-inclusion) of sets of inserted/deleted tuples.

Appendix B Proofs of Results

Proof of *Proposition 3.1*:

Let D_0 be an empty instance for the schema $\mathcal{S}(\mathcal{N}(P))$. By being empty, D_0 satisfies $\bigcup_{\mathtt{Q} \in \mathcal{N}(\mathtt{P})} \Sigma(\mathtt{P}, \mathtt{Q})$ (condition (i) for a neighborhood solution). Also, since all the trust relationships are of the "same" kind, condition (ii) on neighborhood solutions is satisfied by D_0. Then, either D_0 is a neighborhood solution, or there exists a neighborhood solution D'' such that $D'' \preceq_{\bar{D}} D_0$. □

Proof of *Corollary 3.1*:

All we need is notice that the possibly inconsistent sink peers in the accessibility graph always have local repairs under the kind of DECS considered (ICs in that case). A solution for a peer P can then be obtained by recursively propagating back neighborhood solutions (that always exist by Proposition 3.1) for peers along the paths that contain P. □

Proof of *Proposition 5.1*:

Membership of *coNP* is established by directly appealing to Definition 3.1 of neighborhood solution. In fact, given neighborhood instance J, after having checked (in polynomial time) if $J \subseteq r\text{-}Chase^{null}(\bar{D}, \Sigma^{-}(\mathtt{P}))$, a non-deterministic algorithm to test that J is *not* a neighborhood solution for P and the neighborhood instance $\bar{D}$ checks if one of the following holds:

1. $J \not\models \Sigma(\mathtt{P})$.
2. $J \downharpoonright \{R\} \neq \bar{D} \downharpoonright \{R\}$, for some $\mathtt{Q} \in \mathcal{N}(\mathtt{P})$ and predicate $R \in \mathcal{S}(\mathtt{Q})$ with $(\mathtt{P}, less, \mathtt{Q}) \in Trust$.
3. There is an instance J' for $\mathcal{S}(\mathcal{N}(\mathtt{P}))$ (the non-deterministic choice) that satisfies conditions (i) and (ii) of Definition 3.1, but $J' <_{\bar{D}}^{\Sigma(\mathtt{P})} J$.

These conditions are the basis for a non-deterministic algorithm: First conditions 1. and 2. can be checked deterministically in polynomial time. If they are passed by J (i.e. the answer is negative), then an instance J' with size polynomially bounded by the size of J is guessed. Next, conditions 1.-2. are checked for J', and 3., for the pair (J, J'). The three tests can be performed in polynomial time in $|J| + |\bar{D}|$. If the answer to any of the tests is *yes*, J is not a neighborhood solution.

Hardness can be proved by reduction of satisfiability of propositional formulas in CNF, which is *coNP*-complete. The reduction is a modification of that used in

Theorem 4.4 of (Chomicki and Marcinkowski 2005) to prove that repairs obtained through deletions are *coNP*-complete. In our case we have to deal with trust relationships and the possible insertion of tuples with *null*. Actually, in our proof the former will be used to exclude the latter.

We now show that the satisfiability of a propositional formula $\varphi : \ \varphi_1 \wedge \varphi_2 \wedge \ldots \varphi_m$ in CNF (i.e. the φ_i are clauses) can be reduced to checking if a particular neighborhood instance is a neighborhood solution for a given peer.

Consider the fixed PDES schema (it does not depend on φ): $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, \mathit{Trust}\rangle$, with $\mathcal{P} = \{\mathtt{P1}, \mathtt{P2}\}$, $\mathfrak{S} = \{S(\mathtt{P1}), S(\mathtt{P2})\}$, $S(\mathtt{P1}) = \{R^1(\cdot,\cdot,\cdot,\cdot)\}$, $S(\mathtt{P2}) = \{R^2(\cdot,\cdot,\cdot,\cdot)\}$, $\mathit{Trust} = \{(\mathtt{P1}, \mathit{same}, \mathtt{P1}), (\mathtt{P1}, \mathit{less}, \mathtt{P2})\}$, and $\Sigma = \{\Sigma(\mathtt{P1},\mathtt{P1}), \Sigma(\mathtt{P1},\mathtt{P2})\}$, with:

$$\begin{aligned}\Sigma(\mathtt{P1},\mathtt{P1}) = \{&\forall x_1 y_1 y_2 z_1 z_2 w_1 w_2 (R^1(x_1,y_1,z_1,w_1) \wedge R^1(x_1,y_2,z_2,w_2) \rightarrow y_1 = y_2),\\ &\forall x_1 y_1 z_1 w_1 (R^1(x_1,y_1,z_1,w_1) \rightarrow \exists x_2 y_2 z_2 R^1(x_2,y_2,z_2,z_1))\}.\\ \Sigma(\mathtt{P1},\mathtt{P2}) = \{&\forall xyzw(R^1(x,y,z,w) \rightarrow R^2(x,y,z,w))\}.\end{aligned}$$

Now, consider a propositional formula φ as above, on which the instances for the peer system will depend. Let $\mathfrak{D}_\varphi = \{D(\mathtt{P1}), D(\mathtt{P2})\}$ be the instance for $\mathfrak{P}$, with:

$$\begin{aligned} D(\mathtt{P1}) &= \{R^1(p_j, 0, \varphi_i, \varphi_{i+1}) \mid p_j \text{ occurs negatively in } \varphi_i\} \cup \\ &\quad\ \{R^1(p_j, 1, \varphi_i, \varphi_{i+1}) \mid p_j \text{ occurs positively in } \varphi_i\}\cdot \\ D(\mathtt{P2}) &= \{R^2(a,b,c,d) \mid R^1(a,b,c,d) \in D(\mathtt{P1})\}\cdot \end{aligned}$$

(The addition $i{+}1$ is meant to be modulo the number m of clauses in ϕ.) Notice that tables R^1 and R^2 for peers P1 and P2, respectively, have the same rows. Intuitively, the UDEC in $\Sigma(\mathtt{P1},\mathtt{P1})$ ensures that, for every proposition in the first attribute of R^1, the truth assignment, if any, is unique; whereas the RDEC in it, ensures that there are assignments that make all clauses in the formula true.

We now show that the neighborhood instance $\bar{D}$, with the empty relation for R^1 plus the original contents of $D(\mathtt{P2})$, is the neighborhood solution for P1 with initial neighborhood instance $D(\mathtt{P1}) \cup D(\mathtt{P2})$ if and only if φ is *not* satisfiable. In this case, $\bar{D}$ would be obtained through the deletion of all tuples from R^1 in $D(\mathtt{P1})$. Notice that due to the trust relations and the DEC in $\Sigma(\mathtt{P1},\mathtt{P2})$, only tuple deletions from peer P1's instance are admissible updates.

To prove that $\bar{D}$ being a neighborhood solution for P1 implies that φ is not satisfiable, assume by contradiction that φ is satisfiable. Then, there is an assignment σ that makes φ true.

The instance $\bar{D}' := \{R^1(p, 0, \varphi_i, \varphi_{i+1}) \in D(\mathtt{P1}) \mid \sigma(p) = 0\} \cup \{R^1(p, 1, \varphi_i, \varphi_{i+1}) \in D(\mathtt{P1}) \mid \sigma(p) = 1\} \cup D(\mathtt{P2})$ is a subinstance of $D(\mathtt{P1}) \cup D(\mathtt{P2})$, $\bar{D}' \downharpoonright \mathcal{S}(\mathtt{P1}) \neq \emptyset$, satisfies the DECs, and does not modify the more trusted relations, i.e. $\bar{D}' \downharpoonright \mathcal{S}(\mathtt{P2}) = D(\mathtt{P2})$. Thus, $\bar{D}$ cannot be a neighborhood solution since $\bar{D}' <^{\Sigma(\mathtt{P1})}_{D(\mathtt{P1}) \cup D(\mathtt{P2})} \bar{D}$.

Now we show that if φ is not satisfiable, then $\bar{D}$ is a neighborhood solution for P1 when starting with neighborhood instance $D(\mathtt{P1}) \cup D(\mathtt{P2})$. Assume by contradiction that $\bar{D}$ is not a neighborhood solution. Since $\bar{D}$ satisfies all the DECs and respects the trust relationships, $\bar{D}$ cannot be a neighborhood solution only if there is a neighborhood instance $\bar{D}'$, such that: $\bar{D}' \models \Sigma(\mathtt{P1})$; $\bar{D}' \downharpoonright S(\mathtt{P2}) = D(\mathtt{P2})$, and $\bar{D}' <^{\Sigma(\mathtt{P1})}_{D(\mathtt{P1}) \cup D(\mathtt{P2})} \bar{D}$. Since $\bar{D}'$ can be obtained only through tuple deletions, it holds:

$\bar{D} \subsetneqq D' \subseteq D(\texttt{P1}) \cup D(\texttt{P2})$. Thus, there is at least one tuple $R^1(\bar{t}) \in (\bar{D}' \cap D(\texttt{P1}))$. Due to the UDEC in $\Sigma(\texttt{P1},\texttt{P2})$, we conclude that, for every $i \in [1,m]$, there exists a p and v with $R^1(p,v,\varphi_i,\varphi_{i+1}) \in (D' \cap D(\texttt{P1}))$. Using these tuples we can define the following assignment σ':

$$\sigma'(p) = \begin{cases} 1 & \text{if } R^1(p,1,\varphi_i,\varphi_{i+1}) \in D' \text{ with } i \in [1,m] \\ 0 & \text{if } R^1(p,0,\varphi_i,\varphi_{i+1}) \in D' \text{ with } i \in [1,m] \end{cases}$$

The assignment is well defined, because the functional dependency in $\Sigma(\texttt{P1},\texttt{P1})$ ensures that only one value exists for each proposition. By construction, σ' is an assignment that satisfies φ. We have reached a contradiction, which completes the proof. $\square$

PROOF OF *Proposition 5.2*:

First we prove membership of Π^P_2. An atom $R(\bar{t}) \in \mathit{localCore}(\texttt{P},\bar{D})$ if for every $D' \in \mathit{NS}(\texttt{P},\bar{D})$, $D' \models R(\bar{t})$. Thus, a non-deterministic algorithm that checks if $R(\bar{t}) \notin \mathit{localCore}(\texttt{P},\bar{D})$ guesses an instance J of $\mathcal{S}(\mathcal{N}(\texttt{P}))$, next checks if it is a neighborhood solution for $\texttt{P}$ and $\bar{D}$, and finally, if $R(\bar{t}) \notin J$. By Proposition 5.1, the first of these two tests is in *coNP*; and the second one is in polynomial time. Thus, the problem is in Π^P_2.

Hardness holds by a reduction from satisfiability of a quantified propositional formulas (QBF) β of the form $\forall p_1 \cdots \forall p_k \exists q_1 \cdots \exists q_l \psi$, where ψ is a quantifier-free propositional formula in CNF, i.e. of the form $\psi_1 \wedge \ldots \wedge \psi_m$, where the ψ_i are clauses. This problem is Π^P_2-complete (Schaefer and Umans 2008; Papadimitriou 1994). (The reduction is adapted from that for Theorem 4.7 in (Chomicki and Marcinkowski 2005).)

We construct a PDES schema (independent from β) $\mathfrak{P}_0 = \langle \mathcal{P}_0, \mathfrak{S}_0, \Sigma_0, \mathit{Trust}_0 \rangle$, with $\mathcal{P}_0 = \{\texttt{P1},\texttt{P2}\}$, $\mathfrak{S}_0 = \{\mathcal{S}(\texttt{P1}), \mathcal{S}(\texttt{P1})\}$, $\mathcal{S}(\texttt{P1}) = \{R(\cdot,\cdot,\cdot), T(\cdot)\}$, $\mathcal{S}(\texttt{P2}) = \{\mathit{Clause}(\cdot), \mathit{Var}(\cdot)\}$, $\mathit{Trust}_0 = \{(\texttt{P1}, \mathit{less}, \texttt{P2})\}$, and the DECs $\Sigma_0 = \{\Sigma(\texttt{P1},\texttt{P2}), \Sigma(\texttt{P1},\texttt{P1})\}$ with:

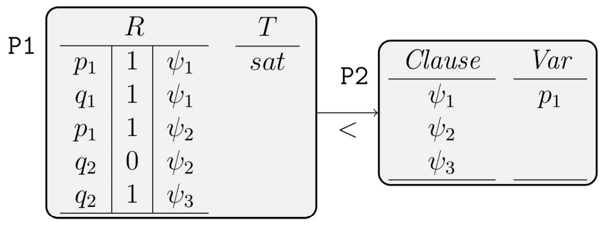
$$\begin{aligned} \Sigma(\texttt{P1},\texttt{P2}) \;&=\; \{\forall x(\mathit{Clause}(x) \rightarrow \exists yz R(y,z,x)), \\ &\qquad \forall x(\mathit{Var}(x) \rightarrow R(x,1,a) \vee R(x,0,a))\}, \\ \Sigma(\texttt{P1},\texttt{P1}) \;&=\; \{\forall x y_1 y_2 z_1 z_2 (R(x,y_1,z_1) \wedge R(x,y_2,z_2)) \rightarrow y_1 = y_2, \\ &\quad \forall xyzw(R(x,y,z) \wedge T(w) \rightarrow w \neq \mathbf{sat} \vee \mathit{IsNotNull}(x) \vee \mathit{IsNotNull}(y))\}\cdot \end{aligned}$$

Now, given a QBF β, we construct an instance $\bar{D}_\beta$ for the neighborhood schema $\mathcal{S}(\mathcal{N}(\texttt{P1}))$ around $\texttt{P1}$, such that: $T(\mathbf{sat}) \in \mathit{localCore}(\texttt{P1},\bar{D}_\beta)$ iff β is true.

Now, for $\beta = \forall p_1 \cdots \forall p_k \exists q_1 \cdots \exists q_l (\psi_1 \wedge \ldots \wedge \psi_m)$, $\bar{D}_\beta := D_\beta(\texttt{P1}) \cup D_\beta(\texttt{P2})$, with:

$$\begin{aligned} D_\beta(\texttt{P1}) \;&:=\; \{R(\mathit{var}_j,1,\psi_i) \mid \mathit{var}_j \text{ occurs positively in } \psi_i\} \cup \\ &\qquad \{R(\mathit{var}_j,0,\psi_i) \mid \mathit{var}_j \text{ occurs negatively in } \psi_i\} \cup \\ &\qquad \{\mathit{Var}(p_i) \mid p_i \text{ is universally quantified in } \psi\} \cup \{T(\mathbf{sat})\}\cdot \\ D_\beta(\texttt{P2}) \;&:=\; \{\mathit{Clause}(\psi_1), \ldots, \mathit{Clause}(\psi_m)\}\cdot \end{aligned}$$

Intuitively, relation $R(x,y,z)$ is used to provide a truth value y to variable x in conjunct z. This truth value will be unique across ψ due to the integrity constraints



R			T
p_1	1	ψ_1	*sat*
q_1	1	ψ_1	
p_1	1	ψ_2	
q_2	0	ψ_2	
q_2	1	ψ_3	

Clause	*Var*
ψ_1	p_1
ψ_2	
ψ_3	

Fig. B 1. Instances for a peer system

on R contained in $\Sigma(\texttt{P1},\texttt{P1})$. The first DEC in $\Sigma(\texttt{P1},\texttt{P2})$ ensures that, for every clause ψ_i, there is, if possible, a literal which is true in it. If it is not possible (the formula is not true), it inserts a tuple of the form $R(null, null, \psi_i)$. The second DEC ensures that all possible assignments for the universally quantifies variables are tested in different solutions. It uses a constant, a, which is different from all ψ_i. The first IC in $\Sigma(\texttt{P1},\texttt{P1})$ enforces that, in each solution, each propositional variable takes a unique value. Finally, the second IC in $\Sigma(\texttt{P1},\texttt{P1})$ ensures that if $R(null, null, \psi_i)$ is true, then predicate $T(\textbf{sat})$ should not be part of the neighborhood solution. In this way, formula β is true if and only of $T(\textbf{sat}) \in localCore(\texttt{P1}, \bar{D}_\beta)$. To conclude the proof, we illustrate the reduction with the following example.

Example 7
Consider the QBF $\forall p_1 \exists q_1 \exists q_2(\psi_1 \wedge \psi_2 \wedge \psi_3)$, with $\psi_1\colon\ (p_1 \vee q_1)$, $\psi_2\colon\ (p_1 \vee \neg q_2)$, and $\psi_3\colon\ q_2$. Instance $\bar{D}_\beta$ is the union of the instances in Figure B 1.

On this basis, the neighborhood solutions for P1 and $\bar{D}_\beta$ are:
$D_1 = \{R(p_1, 1, a), R(p_1, 1, \psi_1), R(q_1, 1, \psi_1), R(p_1, 1, \psi_2), R(q_2, 1, \psi_3), T(\textbf{sat})\} \cup D(\texttt{P2})$,
$D_2 = \{R(p_1, 0, a), R(q_1, 1, \psi_1), R(q_2, 1, \psi_3), R(null, null, \psi_2)\} \cup D(\texttt{P2})$, and
$D_3 = \{R(p_1, 0, a), R(q_1, 1, \psi_1), R(q_2, 0, \psi_2), R(null, null, \psi_3)\} \cup D(\texttt{P2})$.

Since $T(\textbf{sat}) \notin localCore(\texttt{P1}, \bar{D}_\beta) := (D_1 \cap D_2 \cap D_3)\downharpoonright\mathcal{S}(\texttt{P1})$, the QBF formula is false.

□

Proof of *Corollary 5.2*:

Membership is established with a test similar to that in the proof of Proposition 5.2. Hardness follows from Proposition 5.2, because it is about a particular kind of conjunctive queries, namely atomic of the form $\mathcal{Q}(\bar{x})\colon R(\bar{x})$, where R is a predicate for a peer P. The peer consistent answers to this query are exactly the $\bar{c}$s, such that $R(\bar{c})$ is in the local core of P. □

Proof of *Proposition 5.3*:

The existence and uniqueness is straightforward since there are no local restrictions (existence), and there is no non-determinism involved (uniqueness). The unique solution for a peer P can be computed by means of Algorithm ImportSolution shown in Figure B 2. It recursively computes the solutions for all the peers that are accessible from P. The base case occurs when a peer Q has no DECs (line 4). In that case,

```
1  Algorithm: ImportSolution
2  Input: An instance $\mathfrak{D}$ for a PDES schema $\mathfrak{P} = \langle \mathcal{P}, \mathfrak{S}, \Sigma, Trust \rangle$ and a peer $\mathtt{P} \in \mathcal{P}$
3  Output: The unique solution of $\mathtt{P}$
4  if $\mathtt{P}$ has no outgoing edges then return $D(\mathtt{P})$;
5  else
6      foreach $\mathtt{Q} \in \mathcal{N}^\circ(\mathtt{P})$ do
7          $Sol(\mathtt{Q}, \mathfrak{D}) \leftarrow$ ImportSolution($\mathcal{P}, \mathtt{Q}$);
8      $D' \leftarrow D(\mathtt{P}) \cup \bigcup_{\mathtt{Q} \in \mathcal{N}^\circ(\mathtt{P})} Sol(\mathtt{Q}, \mathfrak{D})$;
9      $NS \leftarrow$ minimal model of Datalog import program $\mathfrak{I}(\mathtt{P}, D')$ ;
10     return $NS\restriction S(\mathtt{P})$;
```

Fig. B 2. Computing the solution for a peer in the import case

its unique solution is its own database $D(\mathtt{Q})$. Otherwise (lines 5-10), the algorithm recursively requests the solutions of its neighbors (lines 6-7) and uses them to construct instance D' (line 8). Then, the unique neighborhood solution for the peer consists of the minimal model of $\mathfrak{I}(\mathtt{P}, D')$ (line 9). By restricting $\mathtt{P}$'s neighborhood solution to $\mathtt{P}$'s schema we get $\mathtt{P}$'s solution (line 10). □

Proof of *Proposition 6.1*:

We will prove this result for the case where the central peer trusts its neighbors as much as itself, which is more general in some sense than that where it trusts neighbors' data more, because more alternatives for repairs come up, using the full power of disjunctive programs. Below D is $\mathtt{P}$'s neighborhood instance for which neighborhood solutions are specified by means of the program in Definition 6.1. $\mathcal{C}$ is the set of constraints, i.e. UDECs and RDECs, for the neighborhood.

Given the trust assumptions, the program takes a special form, as follows. For exchange constraints in $\mathcal{C}$ of the forms:

(a) *Universal constraint* (UDEC):
$$\forall \bar{x} (\bigwedge_{i=1}^{m} P_i(\bar{x}_i) \longrightarrow \bigvee_{j=1}^{n} Q_j(\bar{y}_j) \vee \varphi). \qquad (B1)$$

(b) *Referential constraint*: (RDEC)
$$\forall \bar{x} \, (P(\bar{x}) \longrightarrow \exists \bar{y} \, Q(\bar{x}', \bar{y})) \; .^{27} \qquad (B2)$$

the neighborhood solution program $\Pi(\mathtt{P}, D)$ becomes:

1. $dom(c)$ for every $c \in Adom(D) \smallsetminus \{null\}$.
2. The fact $P(\bar{a})$ for every atom $P(\bar{a}) \in D$.
3. For every UDEC ψ of the form (B1), the rule:
$$\bigvee_{i=1}^{n} P_{i\_}(\bar{x}_i, \mathbf{f}) \vee \bigvee_{j=1}^{m} Q_{j\_}(\bar{y}_j, \mathbf{t}) \longleftarrow \bigwedge_{i=1}^{n} P_{i\_}(\bar{x}_i, \mathbf{t}^\star), \bigwedge_{j=1}^{m} Q_{j\_}(\bar{y}_j, \mathbf{f}^\star), \bigwedge_{x_l \in RelV(\psi)} dom(x_l), \bar{\varphi}.$$
where $RelV(\psi)$ is the set of relevant attributes for ψ, $\bar{x} = \bigcup_{i=1}^{n} x_i$, and $\bar{\varphi}$ is a conjunction of built-ins that is equivalent to the negation of φ.
4. For every RDEC ψ of the form (B2), the rules:[28]

[28] Literal $dom(\bar{x})$ denotes the conjunction of the atoms $dom(x_j)$ for $x_j \in \bar{x}$.

$P\_(\bar{x}, \mathbf{f}) \vee Q\_(\bar{x}', \overline{null}, \mathbf{t}) \leftarrow P\_(\bar{x}, \mathbf{t}^\star), \mathit{not}\ aux_\psi(\bar{x}'),\ dom(\bar{x}')\cdot$
And for every $y_i \in \bar{y}$:
$aux_\psi(\bar{x}') \leftarrow Q\_(\bar{x}', \bar{y}, \mathbf{t}^\star),\ \mathit{not}\ Q\_(\bar{x}', \bar{y}, \mathbf{f}),\ dom(\bar{x}'),\ dom(y_i).$
$aux_\psi(\bar{x}') \leftarrow Q(\bar{x}', \overline{null}),\ \mathit{not}\ Q\_(\bar{x}', \overline{null}, \mathbf{f}),\ dom(\bar{x}').$

5. For every predicate $P \in \mathcal{S}(\mathcal{N}(\mathtt{P}))$, the rules:
$P\_(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x})\cdot \quad P\_(\bar{x}, \mathbf{t}^\star) \leftarrow P\_(\bar{x}, \mathbf{t})\cdot$
$P\_(\bar{x}, \mathbf{f}^\star) \leftarrow P(\bar{x}, \mathbf{f})\cdot \quad P\_(\bar{x}, \mathbf{f}^\star) \leftarrow dom(\bar{x}), \mathit{not}\ P(\bar{x})\cdot$
6. For every predicate $P \in \mathcal{S}(\mathcal{N}(\mathtt{P}))$, the *interpretation rules*:
$P\_(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P\_(\bar{x}, \mathbf{t})\cdot \quad P\_(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{x}), \mathit{not}\ P\_(\bar{x}, \mathbf{f})\cdot$
7. For every predicate $P \in \mathcal{S}(\mathcal{N}(\mathtt{P}))$, the *coherence constraints*:
$\leftarrow \quad P\_(\bar{x}, \mathbf{t}),\ P\_(\bar{x}, \mathbf{f})\cdot$

The claim is: If $\mathcal{M}$ is a stable model of $\Pi(\mathtt{P}, D)$, then D_M is a neighborhood solution repair of D. Furthermore, the neighborhood solutions obtained in this way are all the neighborhood solutions of D. We recall that for a stable model of $\Pi(\mathtt{P}, D)$,

$$D_M = \{P(\bar{a}) \ \mid\ P \in \mathcal{S}(\mathcal{N}(\mathtt{P})) \text{ and } P\_(\bar{a}, \mathbf{t}^{\star\star}) \in M\}\cdot \qquad \text{(B3)}$$

The proof follows directly from Propositions 1 and 2 below, which require in their turn some lemmas and intermediate definitions. □

In the following, for a disjunctive program Π and a set of ground atoms M, Π^M is the positive ground program obtained by the Gelfond-Lifschitz transformation (Gelfond and Lifschitz 1991):

$$\Pi^M = \{H \leftarrow B \ \mid\ H \leftarrow B, \mathit{not} A_1, \ldots, \mathit{not} A_m \in \mathit{ground}(\Pi), \text{ and } A_1, \ldots A_m \notin M\}\cdot$$

Lemma 1
Given an instance D and a RDEC-acyclic set of UDECs and RDECs, if M is a stable model of $\Pi(\mathtt{P}, D)$, then exactly one of the following cases holds:

1. $P(\bar{a})$, $P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{t}^{\star\star})$ belong to M, and no other $P\_(\bar{a}, v)$, for v an annotation, belongs to M.
2. $P(\bar{a})$, $P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{f})$ belong to M, and no other $P\_(\bar{a}, v)$, for v an annotation, belongs to M.
3. $P(\bar{a}) \notin M$, and $P\_(\bar{a}, \mathbf{t})$, $P\_(\bar{a}, \mathbf{t}^\star)$, $P\_(\bar{a}, \mathbf{t}^{\star\star})$ belong to M, and no other $P\_(\bar{a}, v)$, for v an annotation, belongs to M.
4. $P(\bar{a}) \notin M$, and no $P\_(\bar{a}, v)$, for v an annotation, belongs to M.

Proof: For an atom $P(\bar{a})$, there are two possibilities:

(a) $P(\bar{a}) \in M$. Then, from rule 5., $P\_(\bar{a}, \mathbf{t}^\star) \in M$. Two cases are possible now: $P\_(\bar{a}, \mathbf{f}) \notin M$ or $P\_(\bar{a}, \mathbf{f}) \in M$. In the first case, since M is minimal, $P\_(\bar{a}, \mathbf{t}) \notin M$ and $P\_(\bar{a}, \mathbf{t}^{\star\star}) \in M$. In the second case, due to rule 7., $P\_(\bar{a}, \mathbf{t}) \notin M$. These cases cover the first two in the statement of the lemma.
(b) $P(\bar{a}) \notin M$. Two cases are possible now: $P\_(\bar{a}, \mathbf{t}) \in M$ or $P\_(\bar{a}, \mathbf{t}) \notin M$. In the first one, it also holds $P\_(\bar{a}, \mathbf{t}^{\star\star})$, $P\_(\bar{a}, \mathbf{t}^\star) \in M$, by rules 5. and 6.; and $P\_(\bar{a}, \mathbf{f}) \notin M$ by rule 7. In the second case, $P\_(\bar{a}, \mathbf{t}^\star) \notin M$ (because M is minimal), $P\_(\bar{a}, \mathbf{f}) \notin M$ (because $P\_(\bar{a}, \mathbf{t}^\star) \notin M$, and M is minimal). These cases cover the last two in the statement of the lemma.

□

From two database instances we can define a structure.

Definition 1
For two database instances D_1 and D_2 over the same schema and domain and a set of constraints $\mathcal{C}$, $M^\star_\mathcal{C}(D_1, D_2)$ is the Herbrand structure $\langle \mathcal{U}, I_\mathcal{P},\ I_\mathcal{B}\rangle$, where $\mathcal{U}$ is the underlying domain,[29] and I_P, I_B are the interpretations for the database predicates (extended with annotation arguments), and the built-ins, respectively. I_P is inductively defined as follows:

1. If $P(\bar{a}) \in D_1$ and $P(\bar{a}) \in D_2$, then $P(\bar{a})$, $P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{t}^{\star\star}) \in I_P$.
2. If $P(\bar{a}) \in D_1$ and $P(\bar{a}) \notin D_2$, then $P(\bar{a})$, $P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{f}) \in I_P$.
3. If $P(\bar{a}) \notin D_1$ and $P(\bar{a}) \notin D_2$, then $P\_(\bar{a}, v) \notin I_\mathcal{P}$ for every annotation v.
4. If $P(\bar{a}) \notin D_1$ and $P(\bar{a}) \in D_2$, then $P\_(\bar{a}, \mathbf{t})$, $P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{t}^{\star\star}) \in I_\mathcal{P}$.
5. For every RDEC $\psi \in \mathcal{C}$ of the form $\forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{y}\, Q(\bar{x}', \bar{y}))$: If $P\_(\bar{a}, \mathbf{t}^{\star\star}) \in I_\mathcal{P}$ and $Q\_(\bar{a}', \bar{b}, \mathbf{t}^{\star\star}) \in I_\mathcal{P}$, with $\bar{a} \neq \mathit{null}$ and at least one $b \in \bar{b}, b \neq \mathit{null}$, then $\mathit{aux}_\psi(\bar{a}') \in I_\mathcal{P}$.

The interpretation $I_\mathcal{B}$ is defined as expected: if Q is a built-in, then $Q(\bar{a}) \in I_\mathcal{B}$ iff $Q(\bar{a})$ is true in classical logic, and $Q(\bar{a}) \notin I_\mathcal{B}$ iff $Q(\bar{a})$ is false. □

Notice that the database instance associated to $M^\star_\mathcal{C}(D_1, D_2)$ through (B3) corresponds exactly to D_2, i.e. $D_{M^\star_\mathcal{C}(D_1,D_2)} = D_2$.

Lemma 2
Given an instance D and a set $\mathcal{C}$ of UDECs and RDECs, if $D' \models_N \mathcal{C}$, then there is a model M of the program $\Pi(\mathtt{P}, D)^M$ with $D_M = D'$. Actually, $M^\star_\mathcal{C}(D, D')$ is such a model.

Proof: Since $D_{M^\star_\mathcal{C}(D,D')} = D'$, we only need to show that $M^\star_\mathcal{C}(D, D')$ satisfies all the rules of $\Pi(\mathtt{P}, D)^{M^\star_\mathcal{C}(D,D')}$. First, by construction, it is clear that rules 2., 5. and 6. are satisfied by $M^\star_\mathcal{C}(D, D')$.

For every UDEC in $\mathcal{C}$, the program has the rule in 3. If its body is satisfied, then the atoms $P_{i}\_(\bar{a}_i, \mathbf{t}^\star) \in M^\star_\mathcal{C}(D, D')$ and $Q_{i}\_(\bar{b}_i, \mathbf{f}) \in M^\star_\mathcal{C}(D, D')$ or $Q_i(\bar{b}_i) \notin M^\star_\mathcal{C}(D, D')$. Also, since the constraint is satisfied, at least one of the $P_i(\bar{a}_i)$ is not in D' or one of the $Q_i(\bar{b}_i)$ is in D'. By construction of $M^\star_\mathcal{C}(D, D')$, at least one of $P_{i}\_(\bar{a}_i, \mathbf{f})$ or $Q_{i}\_(\bar{b}_i, \mathbf{t})$ is in $M^\star_\mathcal{C}(D, D')$. Therefore, the head of the rule is also satisfied.

For every RDEC in $\mathcal{C}$, there are the rules 4. By construction of $M^\star_\mathcal{C}(D, D')$, for every $\psi \in \mathcal{C}$, those that define $\mathit{aux}_\psi(\bar{x})$ are satisfied.

If the body of the first rule in 4 is true in $M^\star_\mathcal{C}(D, D')$, it means that the constraint is not satisfied in the initial instance or at some point along the repair process. Since the constraint is satisfied by D', the satisfaction is restored by adding $Q\_(\bar{x}, \overline{\mathit{null}})$ or by deleting $P(\bar{x})$. This implies that $Q\_(\bar{x}, \overline{\mathit{null}}, \mathbf{t}) \in M^\star_\mathcal{C}(D, D')$ or $P\_(\bar{x}, \mathbf{f}) \in M^\star_\mathcal{C}(D, D')$. As a consequence, the first (or second) rule is satisfied. Then,

[29] In this case it can be restricted to the active domain of the neighborhood instance D (or the union of the active domains of D_1, D_2) plus the constant *null*.

by construction of $M^\star_{\mathcal{C}}(D, D')$, $P\_(\bar{a}, \mathbf{f}) \in M^\star_{\mathcal{C}}(D, D')$, and the head of the rule is satisfied. □

The next lemma shows that if M is a minimal model of the program $\Pi(\mathtt{P}, D)^M$, then D_M satisfies the constraints.

Lemma 3
Given a database D and a set of constraints $\mathcal{C}$, if M is a stable model of the program $\Pi(\mathtt{P}, D)$, then $D_M \models_N \mathcal{C}$.

Proof: We want to show that $D_M \models_N \psi$, for every constraint $\psi \in \mathcal{C}$. There are two cases to consider:

A. IC ψ is a UDEC. Since M is a model of $\Pi(\mathtt{P}, D)^M$, M satisfies rules 3. of $\Pi(\mathtt{P}, D)$. Then, at least one of the following cases holds:
   (a) $P_i\_(\bar{a}, \mathbf{f}) \in M$. Then, $P_i(\bar{a}, \mathbf{t}^{\star\star}) \notin M$ and $P(\bar{a}) \notin D_M$ (by Lemma 1). Hence, $P_i(\bar{a}) \notin D_M$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_M \models_N \bigwedge_{i=1}^n P_i(\bar{x_i}) \rightarrow \bigvee_{j=1}^m Q_j(\bar{y_j}) \vee \varphi$ holds.
   (b) $Q_j\_(\bar{a}, \mathbf{t}) \in M$. It is symmetric to the previous one.
   (c) It is not true that $M \models_N \bar{\varphi}$. Then $M \models_N \varphi$. Hence, φ is true, and $D_M \models_N \bigwedge_{i=1}^n P_i(\bar{x_i}) \rightarrow \bigvee_{j=1}^m Q_j(\bar{y_j}) \vee \varphi$ holds.
   (d) $P_i\_(\bar{a}, \mathbf{t}^\star) \notin M$. Given that M is minimal, just the last item in Lemma 1 holds. This means $P_i(\bar{a}, \mathbf{t}^{\star\star}) \notin M$, $P_i(\bar{a}) \notin D_M$ and $P_i(\bar{a}) \notin D_M$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_M \models_N \bigwedge_{i=1}^n P_i(\bar{x_i}) \rightarrow \bigvee_{j=1}^m Q_j(\bar{y_j}) \vee \varphi$ holds.
   (e) $Q_j\_(\bar{a}, \mathbf{f}) \notin M$ or $Q_j(\bar{a}) \in M$. Given that M is minimal, just the first item in Lemma 1 holds. Then, $Q_j\_(\bar{a}, \mathbf{t}^{\star\star}) \in M$, $Q_j(\bar{a}) \in D_M$ and $D_M \models_N Q_j(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_M \models_N \bigwedge_{i=1}^n P_i(\bar{x_i}) \rightarrow \bigvee_{j=1}^m Q_j(\bar{y_j}) \vee \varphi$ holds.

B. Formula ψ is a RDEC. Since M is a model of $\Pi(\mathtt{P}, D)^M$, M satisfies rules 4. of $\Pi(\mathtt{P}, D)$. Then, at least one of the following cases holds:
   (a) $P\_(\bar{a}, \mathbf{f}) \in M$. Then, $P_i\_(\bar{a}, \mathbf{t}^{\star\star}) \notin M$ and $P(\bar{a}) \notin D_M$ (by Lemma 1). Hence, $P_i\_(\bar{a}) \notin D_M$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_M \models_N (P(\bar{x}) \rightarrow Q(\bar{x}', y))$ holds.
   (b) $Q\_(\bar{a}', \overline{null}, \mathbf{t}) \in M$. It is symmetric to the previous one.
   (c) $P\_(\bar{a}, \mathbf{t}^\star) \notin M$. Given that M is minimal, just the last item in Lemma 1 holds. This means $P\_(\bar{a}, \mathbf{t}^{\star\star}) \notin M$, $P(\bar{a}) \notin D_M$ and $P(\bar{a}) \notin D_M$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_M \models_N (P(\bar{x}) \rightarrow Q(\bar{x}', y))$ holds.
   (d) $aux_\psi(\bar{a}') \in M$. This means that $P\_(\bar{a}, \mathbf{t}^\star) \in M$ and there exists $\bar{b} \neq \overline{null}$ with $Q\_(\bar{a}', \bar{b}, \mathbf{t}^\star) \in M$, $Q\_(\bar{a}, \mathbf{f}) \notin M$, and then, that $P(\bar{a}) \in D_M$ and $Q(\bar{a}, \bar{b}) \in D_M$. Then, the constraint is satisfied.

□

Lemma 4
Let D and D' be instances over the same schema and domain. If M is a minimal

model of $\Pi(\mathtt{P}, D)^{M^\star_{\mathcal{C}}(D,D')}$ with $M \subsetneqq M^\star_{\mathcal{C}}(D, D')$, then there exists M' that is a minimal model of $\Pi(\mathtt{P}, D)^{M'}$ with $D_{M'} <_D D'$.

Proof: Since M is a minimal model of $\Pi(\mathtt{P}, D)^{M^\star_{\mathcal{C}}(D,D')}$, $P(\bar{a}) \in M$ iff $P(\bar{a}) \in D$. By definition of $M^\star_{\mathcal{C}}(D, D')$ and $M \subsetneqq M^\star_{\mathcal{C}}(D, D')$, the only two ways that both models can differ is that, for some $P(\bar{a}) \in D$, $P\_(\bar{a}, \mathbf{f}) \in M^\star_{\mathcal{C}}(D, D')$ and neither $P(\bar{a})$ nor $P\_(\bar{a}, \mathbf{f})$ belong to M, or for some $P(\bar{a}) \notin D$, $\{P\_(\bar{a}, \mathbf{t}), P(\bar{a}, \mathbf{t}^\star), P\_(\bar{a}, \mathbf{t}^{\star\star})\} \subseteq M^\star_{\mathcal{C}}(D, D')$ and none of $P(\bar{a})$, $P\_(\bar{a}, \mathbf{t})$, $P(\bar{a}, \mathbf{t}^\star)$, $P\_(\bar{a}, \mathbf{t}^{\star\star})$ belong to M. Now, we can use the interpretation rules over M to construct M' that is a minimal model of $\Pi(\mathtt{P}, D)^{M'}$, as follows:

1. If $P(\bar{a}) \in M$ and $P\_(\bar{a}, \mathbf{f}) \notin M$, then $P(\bar{a}), P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{t}^{\star\star}) \in M'$.
2. If $P(\bar{a}) \in M$ and $P\_(\bar{a}, \mathbf{f}) \in M$, then $P(\bar{a}), P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{f}) \in M'$.
3. If $P(\bar{a}) \notin M$ and $P\_(\bar{a}, \mathbf{t}) \notin M$, then nothing is added to M'.
4. If $P(\bar{a}) \notin M$ and $P\_(\bar{a}, \mathbf{t}) \in M$, then $P\_(\bar{a}, \mathbf{t})$, $P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{t}^{\star\star}) \in M'$.

It is clear that M' satisfies the coherence constraints, and is a minimal model of $\Pi(\mathtt{P}, D)^{M'}$.

It just rests to prove that $D_{M'} <_D D'$. First, we prove that $D_{M'} \leq_D D'$. Let us suppose $P(\bar{a}) \in \Delta(D, D_{M'})$. Then, either $P(\bar{a}) \in D$ and $P(\bar{a}) \notin D_{M'}$ or $P(\bar{a}) \notin D$ and $P(\bar{a}) \in D_{M'}$. In the first case, $P(\bar{a})$, $P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{f})$ belong to M'. These atoms are also in M and, given the only two ways in which M and $M^\star_{\mathcal{C}}(D, D')$ can differ, they are also in $M^\star_{\mathcal{C}}(D, D')$. Hence, $P(\bar{a}) \in \Delta(D, D')$. In the second case, $P\_(\bar{a}, \mathbf{t})$ and $P\_(\bar{a}, \mathbf{t}^\star)$ belong to M'. These atoms are also in M and, given the only two ways in which M and $M^\star_{\mathcal{C}}(D, D')$ can differ, they are also in $M^\star_{\mathcal{C}}(D, D')$. Hence, $P(\bar{a}) \in \Delta(D, D')$.

We now prove that $D_{M'} <_D D'$. We know that, for some fact $P(\bar{a})$, $P\_(\bar{a}, \mathbf{t}) \in M^\star_{\mathcal{C}}(D, D')$ and $P\_(\bar{a}, \mathbf{t}) \notin M$, or $P\_(\bar{a}, \mathbf{f}) \in M^\star_{\mathcal{C}}(D, D')$ and $P\_(\bar{a}, \mathbf{f}) \notin M$. If $P\_(\bar{a}, \mathbf{f})$ is in $M^\star_{\mathcal{C}}(D, D')$ and not in M, then, $P(\bar{a}) \in \Delta(D, D')$, but $P(\bar{a}) \notin \Delta(D, D_{M'})$. Alternatively, if $P\_(\bar{a}, \mathbf{t})$ and $P\_(\bar{a}, \mathbf{t}^\star)$ belong to $M^\star_{\mathcal{C}}(D, D')$ but not to M, then $P(\bar{a}) \in \Delta(D, D')$, but $P(\bar{a}) \notin \Delta(D, D_{M'})$. Therefore, $D_{M'} <_D D'$. □

Proposition 1

Given a neighborhood instance D and a set $\mathcal{C}$ of UDECs and RDECs, if D' is a neighborhood solution for D with respect to $\mathcal{C}$, then there is a stable model M of the program $\Pi(\mathtt{P}, D)^M$ with $D_M = D'$. Furthermore, the model M corresponds to $M^\star_{\mathcal{C}}(D, D')$.

Proof: By Lemma 2, $M^\star_{\mathcal{C}}(D, D')$ is a model of $\Pi(\mathtt{P}, D)^{M^\star_{\mathcal{C}}(D,D')}$. We now show that it is minimal. Assume, by contradiction, that there exists a model M of $\Pi(\mathtt{P}, D)^{M^\star_{\mathcal{C}}(D,D')}$ with $M \subsetneqq M^\star_{\mathcal{C}}(D, D')$. We can assume, without loss of generality, that M is a minimal model. Since $M \subsetneqq M^\star_{\mathcal{C}}(D, D')$, the model M contains the atom $P(\bar{a})$ iff $P(\bar{a}) \in D$.

By Lemma 4, there exists model M' such that $D_{M'} <_D D'$ and M' is a minimal model of $\Pi(\mathtt{P}, D)^{M'}$. By Lemma 3, $D_{M'} \models_N \mathcal{C}$. This contradicts that D' is a neighborhood solution. □

Proposition 2

If M is a stable model of $\Pi(\mathtt{P}, D)$, then D_M is a neighborhood solution for D with respect to $\mathcal{C}$.

Proof: From Lemma 3, it holds $D_M \models_N \mathcal{C}$. We have to prove that it is $\leq_D$-*minimal*. Let us suppose there is a neighborhood solution D' (that satisfies $\mathcal{C}$) with $D' <_D D_M$. From Proposition 1, $M^\star_{\mathcal{C}}(D, D')$ is a stable model of $\Pi(\mathtt{P}, D)$ and $D_{M^\star_{\mathcal{C}}(D,D')} = D'$ (we denote it simple with $M^\star$ in the rest of the proof).

If $D' <_D D_M$, there is an atom $P(\bar{a}) \in \Delta(D, D_M)$, with $P(\bar{a}) \notin \Delta(D, D')$, or there is an atom $P(\bar{a}, \bar{b}) \in \Delta(D, D_M)$, with $\bar{a}', \bar{b} \neq null$, and an atom $P(\bar{a}, \overline{null}) \in \Delta(D, D')$. We analyze both cases:

1. $P(\bar{a}) \in \Delta(D, D_M)$ and $P(\bar{a}) \notin \Delta(D, D')$:
   Since $P(\bar{a}) \in \Delta(D, D_M)$, $P\_(\bar{a}, \mathbf{t})$ or $P\_(\bar{a}, \mathbf{f})$ belong to M. By Lemma 1, there are two cases:
   (a) $P(\bar{a})$, $P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{f})$ belong to M, and no other $P\_(\bar{a}, v)$, for v an annotation, belongs to M. $P(\bar{a})$, $P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{t}^{\star\star})$ belong to $M^\star$, and, for any other annotation v, $P\_(\bar{a}, v) \notin M^\star$.
   (b) $P\_(\bar{a}, \mathbf{t})$, $P\_(\bar{a}, \mathbf{t}^\star)$ and $P\_(\bar{a}, \mathbf{t}^{\star\star})$ belong to M, and no other $P\_(\bar{a}, v)$, for v an annotation, belongs to M. No $P\_(\bar{a}, v)$, for v an annotation, belongs to $M^\star$.

   If an atom belongs to a model M_1, e.g. $P\_(\bar{a}, \mathbf{f})$, and there is another model M_2 in which it is not present, then there must be in M_2 an atom annotated with $\mathbf{t}$ or $\mathbf{f}$ in order to satisfy the rule that was satisfied in M_1 by $P\_(\bar{a}, \mathbf{f})$. This implies that $M^\star$ has an atom annotated with $\mathbf{t}$ or $\mathbf{f}$ that does not belong to M. This implies that there is an atom that belongs to $\Delta(D, D')$ and that does not belong to $\Delta(D, D_M)$. We have reached a contradiction, because $\Delta(D, D')$ is a proper subset of $\Delta(D, D_M)$.
2. $P(\bar{a}, \bar{b}) \in \Delta(D, D_M)$ and $P(\bar{a}, \overline{null}) \in \Delta(D, D')$:
   If $P(\bar{a}, \bar{b}) \notin M$, then $P\_(\bar{a}, \bar{b}, \mathbf{t}) \in M$, $P(\bar{a}, \overline{null}) \notin M$, $P\_(\bar{a}, \overline{null}, \mathbf{t}) \notin M$.
   Since $P(\bar{a}, \overline{null}) \in \Delta(D, D')$ and $P(\bar{a}, \overline{null}) \notin M$, $P\_(\bar{a}, \overline{null}, \mathbf{t}) \in M^\star$.
   Since $P\_(\bar{a}, \overline{null}, \mathbf{t}) \in M^\star$, there must be a rule representing a RDEC in $\Pi(D, \mathcal{C})$ such that $P\_(\bar{a}, \overline{null}, \mathbf{t})$ is the only true atom in the head. For that rule to be also satisfied by M, there must be another atom in the head of that rule that is true in M but not in $M^\star$. This means that there is a $P(\bar{b}) \in \Delta(D, D_M)$ and $P(\bar{b}) \notin \Delta(D, D')$, which brings us back to case 1. above. Again we obtain a contradiction.

Therefore, it is not possible to have $D' <_D D_M$; and D_M is a neighborhood solution for D. □

References

Brewka, G. 2004. Complex Preferences for Answer Set Optimization. In *Proc. International conference on Principles of Knowledge Representation and Reasoning*. AAAI Press, 213–223.

Buccafurri, F., Leone, N., and Rullo, P. 2000. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering 12,* 5, 845–860.

Chomicki, J. and Marcinkowski, J. 2005. Minimal-Change Integrity Maintenance using Tuple Deletions. *Information and Computation 197,* 1-2, 90–121.

De Giacomo, G., Lembo, D., Lenzerini, and Rosati, R. 2007. On Reconciling Data Exchange, Data Integration, and Peer Data Management. In *Proc. ACM Symposium on Principles of Database Systems.* ACM Press, 133–142.

Faber, W. and Woltran, S. 2011. Manifold Answer-Set Programs and Their Applications. In *Gelfond Festschrift*, M. Balduccini and T. C. Son, Eds. Springer LNAI 6565, 44–63.

Giannotti, F., Pedreschi, D., Sacca, D., and Zaniolo, C. 1991. Non-Determinism in Deductive Databases. In *Proc. International Conference on Deductive and Object-Oriented Databases.* Springer, LNCS 566, 129–146.

Herzig, A., Lorini, E., Hubner, J., Ben-Naim, J., Castelfranchi, C., Demolombe, R., Longin, D., and Vercouter, L. 2008. Prolegomena for a Logic of Trust and Reputation. In *Proc. Third International Workshop on Normative Multiagent Systems.* 143–157.

Hien Nguyen, G., Chatalic, P., and Rousset, M.-C. 2008. A Probabilistic Trust Model for Semantic Peer-to-Peer Systems. In *Proc. European Conference on Artificial Intelligence.* IOS Press, 881–882.

Schaefer, M. and Umans, Ch. 2008. Completeness in the Polynomial-Time Hierarchy: A Compendium. *SIGACT News.*

Xiong, L. and Liu, L. 2004. PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. *IEEE Transactions of Knowledge and Data Engineering 16,* 7, 843–857.

Yang, B. and Garcia-Molina, H. 2003. Designing a Super-Peer Network. In *Proc. International Conference on Data Engineering.* IEEE Computer Society, 49.