HANDLING INCONSISTENCY IN DATABASES AND DATA INTEGRATION SYSTEMS

by

Loreto Bravo

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of

the requirements for the degree of

DOCTOR OF PHILOSOPHY

Ottawa-Carleton Institute for Computer Science

School of Computer Science

at

CARLETON UNIVERSITY

Ottawa, Ontario

January, 2007

Abstract

For several reasons a database may not satisfy certain integrity constraints (ICs), for example, when it is the result of integrating several independent data sources. However, most likely, information in it is still consistent with the ICs; and could be retrieved when queries are answered. Consistent answers with respect to a set of ICs have been characterized as answers that can be obtained from every possible minimal repair of the database. The goal of this research is to develop methods to retrieve consistent answers for a wide and practical class of constraints and queries from relational databases and from data integration systems. We will put special interest on databases with null values. We will give a semantics of satisfaction of constraints in the presence of *null* that generalizes the one used in commercial DBMS. Since there are interesting connections between the area of consistently querying virtual data integration systems and other areas, like querying incomplete databases, merging inconsistent theories, semantic reconciliation of data, schema mapping, data exchange, and query answering in peer data management systems, the results of this research could also be applied to them. In our research, we explore in more depth the connection with virtual data integration systems and peer data management systems.

To my husband

Leo Ferres

In memory of

Javier Pinto

Acknowledgements

I'd like to thank late professor Dr. Javier Pinto for encouraging me to pursue an academic career and for all his help and advice. This thesis is dedicated to his memory.

I'd also like to thank my supervisor Dr. Leo Bertossi for his wisdom, friendship, and for the holistic scholarship. You are the "biest"! I can not imagine having a better advisor and mentor for my Ph.D.

This research has been supported and funded by various organizations including Fundación Juan Pablo II, Mecesup, CITO/IBM Student Internship Program and Carleton University's School of Computer Science. I thank them all for their confidence in me. I am also grateful to the School of Computer Science at Carleton University and the School of Engineering at the P. Universidad Católica de Chile for providing me a nice work environment. In particular I'd like to thank Doug Howe, Joanne Martin, April Alton, Linda Pfeiffer, Sandy Herbert and Claire Ryan from Carleton University; and Alvaro Campos, Soledad Carrión, Cecilia Venegas, Alda Briceño, Carolina Roa and Inés Chieyssal from Universidad Católica de Chile.

I'd also like to thank Carla Corral, Martin Jones, Ilia Auerbach-Ziogas, Juan Afonso, Eze Glinsky, Victoria Baker, Jennifer Gisseleire, Paloma Bertossi, Gerardo Reynaga, Kamilla Johannsdottir, Naty Villanueva, Mauricio Vines, Mitziah and Dr. Cheifetz for their emotional support and for being my family in Canada. I'm specially in debt to my friend Monica Caniupan. We've shared an office for so many year and she's been there to support me in all the ups and downs of the PhD and personal life. Gracias Monique. I also wish to thank my parents: Loreto Celedón and Rafael Bravo; and my siblings: Rafael, Cecilia and Isabel Margarita. Thanks for making me

feel as if we were in the same city.

Lastly, and most importantly, I'd like to thank my husband Dr. Leo Ferres. You've been such an amazing support over all these years!! Thanks for being there for me, without minding if I was in a good, bad, happy, low, sleepy, irritable, caffeinated, annoying, excited, nervous, cheerful, crazy or jumpy mood.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In databases (DB), integrity constraints (ICs) capture the semantics of the application domain and help maintain the correspondence between that domain and its model provided by the database when updates on the database are performed. However, databases may become inconsistent with respect to a given set of integrity constraints. This may happen due, among others, to the following factors: (1) Certain ICs, e.g., general inclusion dependencies, are not part of the SQL Standard [International Organization for Standardization, 2003] and cannot be expressed/maintained by database management systems (DBMS) such as IBM DB2 [IBM, 2006], Oracle [Oracle, 2005], MySQL [MySQL, 2006], Microsoft SQL Server [Microsoft, 2006], PostgreSQL [PostgreSQL, 2006] and Sybase ACE [Sybase, 2006]. (2) Inconsistency with respect to *soft* or *informational* integrity constraints, i.e., constraints that are declared and not enforced, but used in query answering. (3) Integration, virtual or materialized, of heterogeneous databases; where each DB may be consistent but the integration can be inconsistent. (4) Legacy data on which one wants to impose new semantic constraints. (5) Users' constraints that cannot be checked or maintained.

Example 1.1 Consider a database D with two relations[1] $P(\underline{X}, Y)$ and $R(\underline{U}, V)$, and a constraint that says that every value in Y should also be in V. The database management systems (DBMSs) IBM DB2, Oracle, Microsoft MySQL, SQL Server and PostgreSQL would only be able to declare this constraint if V was the primary key.

---

[1]Underlined attributes correspond to the primary key. For basic concepts related to databases and integrity constraints, see Section 2.1.

DBMSs allow the user to declare "triggers" that can be used to enforce constraints. However, if the trigger is not in place since the creation of the database, as may be the case in legacy data, constraints could be violated. □

Example 1.2 Consider two databases D_1 and D_2 that are consistent with respect to their primary keys:

D_1 :

S_1	$\underline{ID}$	N	P
	636	*Paul*	2910357
	531	*Josh*	3552487

D_2 :

S_2	$\underline{ID}$	N	P
	636	*Ann*	2910357
	313	*Ted*	4586844

If we want to merge the data (virtually or by materializing the new instance) into a new table *Student* with the same primary key, we get:

Student	$\underline{ID}$	N	P
	636	*Paul*	2910357
	636	*Ann*	2910357
	531	*Josh*	3552487
	313	*Ted*	4586844

Even though the sources were consistent, the integration of both sources results in an inconsistent database, and it is not clear which tuple has the wrong information: $Student(636, Paul, 2910357)$ or $Student(636, Ann, 2910357)$?

If we were to materialize table *Student*, then we could try to find out which of the tuples is correct and delete the other one. This process could be very time consuming and in some cases, the information could even be unavailable. As an alternative we could delete both of them. With this option we might delete data that is still useful.

If we are using virtual integration, i.e., table *Student* is not materialized, then in order to restore consistency, we would need to modify the sources. In many cases, we

may not have sufficient permission to do so; and if we do, we would have the same problem as described in the materialized case. □

This example shows that sometimes, even though we are able to identify the set of tuples that are directly involved in inconsistencies, we might not be able to create a consistent database by modifying the database. If we do not have permission to modify the data or if there is not enough information about how to restore the data, we would be unable to solve the inconsistencies. On the other hand, even if we have access to all the information, the process can be complex and nondeterministic, and as a consequence, may demand too much time and resources. Therefore, trying to repair a database can be expensive, impossible or undesirable.

In any of the scenarios above and others, we are in the presence of an inconsistent database, where only a portion of the information is inconsistent with respect to the intended semantics of the database. Even though the database is inconsistent, we might still need or want to use it.

Example 1.3 Consider a student database D with a relation $Student(ID, N, P)$, where ID is the student number, N is the name, and P is the phone number.

Student	ID	N	P
	636	Paul	2910357
	636	Ann	2910357
	531	Josh	3552487
	313	Ted	4586844

The database D does not satisfy the ICs stating that ID is the primary key, since there are two tuples with $ID = 636$. If we delete the data involved in inconsistencies, we might lose information. For example, even though we do not know the name of student with $ID = 636$, we do know that this student's phone number is 2910357. □

A reasonable approach is to define what data is consistent, and to be able to retrieve only the consistent data stored in inconsistent databases. In other words, we could think of ICs on a database as constraints on the answers to queries rather than on the information stored in the database. Then, retrieving answers to queries that are consistent with respect to the ICs becomes a central issue in the development, implementation, and use of DBMSs.

The first notion of consistent answer to a first-order (FO) query was defined in [Arenas *et al.*, 1999], where a computational mechanism for obtaining consistent answers was also presented. Intuitively speaking, a ground tuple $\bar{t}$ to a first-order query $Q(\bar{x})$ is *consistent* in a possibly inconsistent relational database instance D if it is an answer to $Q(\bar{x})$ in every minimal repair of D, that is, in every database instance over the same schema and domain that differs from D by a minimal set (under set inclusion) of inserted or deleted tuples. In other words, the consistent data in an inconsistent database is invariant under sensible forms of restorations of the consistency of the database. The next example will help to clarify these concepts.

Example 1.4 (example 1.3 continued) There are two possible ways of repairing D with respect to the primary key:

D' :

Student	ID	N	P
	636	*Paul*	2910357
	531	*Josh*	3552487
	313	*Ted*	4586844

D'' :

Student	ID	N	P
	636	*Ann*	2910357
	531	*Josh*	3552487
	313	*Ted*	4586844

If we want to know the phone number of the student with $ID = 636$, we can ask a query to get the phone number of the student with $ID = 636$. The answer to the query obtained from the first and second repair is in both cases 2910357. Therefore the consistent answer is 2910357. Although this data is involved in an inconsistency,

useful information can be obtained from it. If we had deleted the data involved in inconsistency, we would have lost this information. If instead we pose ask if there is a student called Paul, the consistent answer would be *No*, since in the second repair there is no student with name Paul. □

A definition of consistent answer has been given in [Arenas *et al.*, 1999] in terms of database repairs. The database repairs are used as auxiliary concepts of the definition, but it does not mean that all the possible repairs need to be produce in order to obtain the consistent answers.

Consistent answers can be sometimes obtained by posing a rewritten query on the database [Arenas *et al.*, 1999]. The rewriting uses the ICs. The rewriting operator was analyzed in [Arenas *et al.*, 1999] in terms of soundness, completeness and termination. Syntactical classes of queries and ICs were identified for which the operator has these properties. The rewriting technique introduced in [Arenas *et al.*, 1999] was successfully implemented, but only for some restricted kind of constraints and queries [Celle and Bertossi, 2000].

The mechanism presented in [Arenas *et al.*, 1999] has some limitations in terms of the ICs and queries it can handle. Although most of the ICs found in database praxis are covered by the positive cases in [Arenas *et al.*, 1999], the queries are restricted to quantifier-free conjunctions of literals.

In [Fuxman and Miller, 2005], a tight class of conjunctive queries was identified for which the query rewriting technique can also be applied. Consistent query answering was implemented for this class of queries [Fuxman *et al.*, 2005a; Fuxman *et al.*, 2005b].

In [Arenas *et al.*, 2000a; Arenas *et al.*, 2003], a more general methodology to retrieve consistent answers, based on logic programs with a stable model semantics was introduced. There is a one-to-one correspondence between the stable models of the logic programs and the database repairs. More general queries could be considered,

but ICs were restricted to be "binary", i.e., universal with at most two database literals (plus built-in formulas). A similar, independent approach to database repair based on logic programs was also presented in [Greco *et al.*, 2001].

In [Arenas *et al.*, 2000b], annotated logic was used in such a way that the database repairs turned out to be certain minimal models of a theory written in annotated logic. This was done with the intention of providing a logical specification of the repairs of the database and developing mechanisms for consistent answering general first-order queries. In [Barceló and Bertossi, 2002; Barceló; and Bertossi, 2003], this theory was transformed into a disjunctive logic program with annotation constants and a stable model semantics. The logic program specification works for the whole class of the so-called universal constraints, in the sense that there is a one-to-one correspondence between repairs and stable models.

The basic idea behind the logic programming based approach to consistent query answering (CQA) is as follows: since we need to reason with *all* the repairs of a database, we had better specify or axiomatize the class of repairs. From a manageable logical specification of this class, different reasoning tasks could be performed, in particular, computation of consistent answers to queries. This approach is able to handle all first-order queries and a much wider class of ICs than a rewriting technique.

CQA from databases with null values has not been considered so far. In most practical cases, we will have to deal with databases with *null*, and, since they cannot be treated as any other constant, their effect on CQA needs to be addressed. In this thesis, we will consider databases with null values, and provide a precise, uniform, and logic-based definition of IC satisfaction in SQL databases with *null* that extends the one used in commercial DBMSs.

We will also use null values to repair databases, in such a way that the problem of retrieving consistent query answers becomes decidable, in contrast to the case where

repairs must appeal to arbitrary, non-null values in the possibly infinite underlying domain.

The results and techniques obtained for obtaining consistent answers from stand alone databases, can be applied also in the context of virtual data integration. In a virtual data integration system (DIS), the user interacts with a mediator that acts as a virtual, global database that integrates data from different sources. These sources are mapped to the global schema. The semantics of the integration system is given by a set of global database instances. Since there is no common and centralized mechanism for maintaining global consistency, the system might violate any global integrity constraints that the global system is expected to satisfy. Even if the sources are consistent, the integration of them might violate the constraints. It becomes crucial to use the ICs at query time to ensure that the system returns only the consistent answers. In this setting, we can use the notion of repairs for stand alone databases for the global instances and, in this way, obtain the consistent answers of the integration system.

There are interesting connections between the area of consistently querying virtual data integration systems and other areas, like querying incomplete databases [van der Meyden, 1998; Grahne, 2002], merging inconsistent theories [Lin and Mendelzon, 1998; Baral *et al.*, 1992], semantic reconciliation of data [Hull, 1997], schema mapping [Rahm and Bernstein, 2001; Doan *et al.*, 2003; Pottinger and Bernstein, 2002], data exchange [Fagin *et al.*, 2003a; Fagin *et al.*, 2003b; Kolaitis *et al.*, 2006], and query answering in *peer-to-peer* (P2P) data exchange systems, or more precisely *peer data management* (PDM) systems [Kementsietsidis *et al.*, 2003; Halevy *et al.*, 2003; Halevy and Madhavan, 2003; Franconi *et al.*, 2004b; Calvanese *et al.*, 2004b; Fuxman *et al.*, 2005c]. We are specially interested in exploring this connection with virtual data PDM systems (see Chapter 7).

A PDM system consists of a set of database peers that share data. A peer has data exchange constraints that relate its data with the data of other peers. A peer might trust more or the same the data owned by the other peers. These trust relationships have not been considered before in the literature, where there was only an implicit assumption that a peer always trusted its own data less than the one of others. We also consider more general types of data exchange constraints. The data exchange constraints can be seen as integrity constraints over a database consisting of the union of the databases in the different peers.

A query is posed on a single peer, which, in order to answer it, may import data from other peers, and modify or delete its own data, in such a way that answers are compatible with the peer's local ICs, the data exchange mappings, and the trust relationships. The techniques and results of CQA need to be adjusted via a redefinition of repairs, in order to take into consideration all these different elements.

One of the differences between a DIS and a PDM system is that in the former case there is a global schema, whereas in the latter this is not the case. Also, in a DIS the queries are posed to the mediator, in opposition to a PDM system where there is no mediator and queries are be posed to the database at a single peer site.

This thesis is structured as follows. In Chapter 2 we introduce some definitions and concepts. Chapter 3 contains the objectives of the thesis. Chapter 4 introduces semantics for IC satisfaction and query answering in databases with null values. Chapter 5 defines and analyzes repair programs and CQA for a very general class of ICs. Chapters 6 and 7 apply the results of the previous chapters to data integration and to PDM systems, respectively. Finally Chapter 8 provides some final conclusions and future work.

Some of the results of this thesis have been published in [Barceló *et al.*, 2003; Bravo and Bertossi, 2003; Bravo and Bertossi, 2004; Bertossi and Bravo, 2004a; Bertossi and

Bravo, 2004b; Bravo and Bertossi, 2005; Bertossi and Bravo, 2005; Bravo and Bertossi, 2006].

Chapter 2

Preliminaries

2.1 Databases and Integrity Constraints

We concentrate on relational databases, and we assume we have a fixed relational schema $\Sigma = (\mathcal{U}, \mathcal{R}, \mathcal{B})$, where $\mathcal{U}$ is the possibly infinite database domain such that $null \in \mathcal{U}$; $\mathcal{R}$ is a fixed set of database predicates (also called relations), where each relation R has an a finite, ordered set of attributes $\mathcal{A}_R$; and $\mathcal{B}$ is a fixed set of built-in predicates, like comparison predicates. The attribute in position i of predicate $R \in \mathcal{R}$ is denoted by $R[i]$. For $A \subseteq \mathcal{A}_R$, $R[A]$ denotes predicate R projected on the attributes in A.

The schema determines a language $\mathcal{L}(\Sigma)$ of first-order predicate logic. A database instance D compatible with Σ can be seen as a finite collection of ground atoms of the form $R(c_1, ..., c_n)$,[1] where R is a predicate in $\mathcal{R}$ and $c_1, ..., c_n$ are constants in $\mathcal{U}$. Built-in predicates have a fixed extension in every database instance, not subject to changes.

A *query* is a first-order formula over language $\mathcal{L}(\Sigma)$. If the query does not have free variables, it is called a *boolean query*. We will later consider more expressive languages such as Datalog queries and its extensions.

Integrity constraints (ICs) can be used to restrict the data stored in the database by imposing semantics on data. The most common type of constraints are functional dependencies and inclusion dependencies.

---

[1] Also called *database tuples*. Finite sequences of constants in $\mathcal{U}$ are simply called *tuples*.

A *functional dependency* (FD) $R : X \to Y$, where X, Y are sets of attributes associated to R, is satisfied in relation R if any two tuples that have the same values in the attributes in X, also have the same values in the attributes in Y. A set of attributes X is a *candidate key* of a relation R if for every attribute Y in R it holds that $R : X \to Y$, and no subset of X has this property. One of the candidate keys can be chosen as the *primary key* (PK).

An *inclusion dependency* (IND) $S[Y] \subseteq R[X]$, where X, Y are sets of attributes from S and R, respectively, is satisfied iff for each tuple s in S, there exists a tuple r in R such that $s[Y] = r[X]$. An inclusion dependency is said to be *full* if $X = \mathcal{A}_R$ and *partial* if $X \subsetneq \mathcal{A}_R$. A *foreign key* is an inclusion dependency $S[Y] \subseteq R[X]$, where X is the primary key of R.

More generally, we can define an *integrity constraint* (IC) as a sentence $\psi \in \mathcal{L}(\Sigma)$ of the form :

$$\forall \bar{x} (\bigwedge_{i=1}^{n} P_i(\bar{x}_i) \longrightarrow \exists \bar{z} (\bigvee_{j=1}^{m} Q_j(\bar{y}_j, \bar{z}_j) \vee \varphi)), \tag{2.1}$$

where $P_i, Q_j \in \mathcal{R}$, $\bar{x} = \bigcup_{i=1}^{m} \bar{x}_i$, $\bar{z} = \bigcup_{j=1}^{n} \bar{z}_j$, $\bar{y}_j \subseteq \bar{x}$, $\bar{x} \cap \bar{z} = \emptyset$, $\bar{z}_i \cap \bar{z}_j = \emptyset$ for $i \neq j$, $\bar{z}_j$ does not have repeated variables for $j = 1, \ldots, n$, and $m \geq 1$. Formula φ is a disjunction of built-in atoms from $\mathcal{B}$, whose variables appear in the antecedent of the implication. Given an IC ψ, $Ant(\psi)$ and $Con(\psi)$ denote the set of predicates in the antecedent and consequent of ψ, respectively.

We will assume that there is a propositional atom **false** $\in \mathcal{B}$ that is always false in a database. Domain constants other than *null* may appear instead of some of the variables in a constraint of the form (2.1). A wide class of ICs can be accommodated in this general syntactic class, e.g. all the ICs used in commercial database management systems. However, the class of tuple-generating-dependencies [Beeri and Vardi, 1984] can not be written with form (2.1), since it would require a conjunction of literals

instead of a disjunction in the consequent of the constraint. We do not consider this type of constraints since, as we will later show, we want to repair inconsistent databases using *null*, and a joint in the consequent could not be enforced by the use of *null*.

ICs of the form (2.1) are safe [Gelder and Topor, 1987; Gelder and Topor, 1991] and then also domain independent [Abiteboul *et al.*, 1995; Ullman, 1988]. This means that the satisfaction of the constraints can be checked by restricting the inspection to the finite *active domain* and the constants appearing in the ICs instead of the whole underlying domain $\mathcal{U}$. The *active domain* consists of all the constants appearing in the extensional relations.

A *universal integrity constraint* (UIC) has the form (2.1), but with $\bar{z} = \emptyset$, i.e., it has no existentially quantified variables:

$$\forall \bar{x} (\bigwedge_{i=1}^{m} P_i(\bar{x}_i) \longrightarrow \bigvee_{j=1}^{n} Q_j(\bar{y}_j) \vee \varphi). \tag{2.2}$$

A *referential integrity constraint* (RIC) is of the form (2.1), but with $m = n = 1$ and $\varphi = \emptyset$, i.e., it is of the form [2]: (here $\bar{x}' \subseteq \bar{x}$ and $P, Q \in \mathcal{R}$)

$$\forall \bar{x} \ (P(\bar{x}) \longrightarrow \exists \bar{y} \ Q(\bar{x}', \bar{y})). \tag{2.3}$$

Class (2.1) includes most ICs commonly found in database practice. Functional dependencies can be expressed by several implications of the form (2.1), each of them with a single equality in the consequent. Partial inclusion dependencies are RICs, and full inclusion dependencies are universal constraints. *Denial constraint* can be expressed as $\forall \bar{x} (\bigwedge_{i=1}^{m} P_i(\bar{x}_i) \longrightarrow \mathbf{false})$. We can also specify *check constraints* (CC) that express conditions on each row in a table. Check constraints can be formulated

---

[2]To simplify the presentation, we are assuming that the existentially quantified variables appear in the last attributes of Q.

with one predicate in the antecedent of (2.1) and only a formula φ in the consequent. For example, $\forall xy(P(x, y) \rightarrow y > 0)$ is a check constraint.

Consider a database D without $null^3$. A database D can be seen as a first-order theory obtained by applying the *domain closure*, *unique names*, and *closed world* assumptions to the original set of ground atoms in D [Reiter, 1984]. The latter assumption makes false any ground atom not explicitly appearing in the set of atoms D. From now on, we denote with $D \models IC$ the fact that the database satisfies IC. In this case, we say that D is consistent with respect to IC; otherwise we say D is inconsistent.

In the following, we will assume that we have a fixed finite set IC of ICs of the form (2.1). Notice that sets of constraints of this form are always consistent in the classical logical sense, because the empty database always satisfies them.

Example 2.1 For $\mathcal{R} = \{P, R, S\}$ and $\mathcal{B} = \{>, =, \textbf{false}\}$, the following are ICs: (a) $\forall xyzw\ (P(x, y) \wedge R(y, z, w) \rightarrow S(x) \vee (z \neq 2 \vee w \leq y))$ (universal); (b)$\forall xy(P(x, y) \rightarrow \exists z\ R(x, y, z))$ (referential); (c)$\forall x(S(x) \rightarrow \exists yz(R(x, y) \vee R(x, y, z)))$. □

Notice that defining φ in (2.1) as a disjunction of built-in atoms is not an important restriction, because an IC that has φ as a more complex formula can be transformed into a set of constraints of the form (2.1). For example, the formula $\forall xy\ (P(x, y) \rightarrow (x > y \vee (x = 3 \wedge y = 8)))$ can be transformed into: $\forall xy\ (P(x, y) \rightarrow (x > y \vee x = 3))$ and $\forall xy\ (P(x, y) \rightarrow (x > y \vee y = 8))$.

Definition 2.1 [Caniupan and Bertossi, 2005] The *dependency graph* $\mathcal{G}(IC)$ for a set of ICs IC of the form (2.1) is defined as follows: Each database predicate P in $\mathcal{R}$ appearing in IC is a vertex, and there is a directed edge (P_i, P_j) from P_i to P_j iff there exists a constraint $\psi \in IC$ such that $P_i \in Ant(\psi)$ and $P_j \in Con(\psi)$. A

---
^{3}We will deal with databases with *null* in Chapter 4
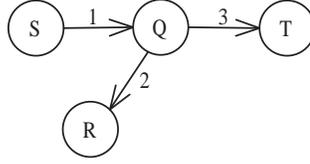
Figure 2.1: Dependency graph of Example 2.2

connected component in a graph is a set-theoretically maximal subgraph such that, for every pair (A, B) of its vertices, there is a path from A to B or from B to A. For a graph $\mathcal{G}$, $\mathcal{C}(\mathcal{G}) := \{c \mid c$ is a connected component in $\mathcal{G}\}$; and $\mathcal{V}(\mathcal{G})$ is the set of vertices of $\mathcal{G}$. □

Example 2.2 For the set IC containing the UICs $\psi_1 \colon S(x) \to Q(x)$ and $\psi_2 \colon Q(x) \to R(x)$, and the RIC $\psi_3 \colon Q(x) \to \exists y T(x, y)$, the dependency graph $\mathcal{G}(IC)$ is shown in Figure 2.1. The edges are labelled just for reference. Edges 1, 2 and 3 correspond to the constraints ψ_1, ψ_2 and ψ_3, respectively. □

Definition 2.2 Given a set IC of UICs and RICs, IC_U denotes the set of UICs in IC. The *contracted dependency graph*, $\mathcal{G}^C(IC)$, of IC is obtained from $\mathcal{G}(IC)$ by replacing, for every $c \in \mathcal{C}(\mathcal{G}(IC_U))$,[4] the vertices in $\mathcal{V}(c)$ by a single vertex and deleting all the edges associated with the elements of IC_U. Finally, IC is said to be *RIC-acyclic* if $\mathcal{G}^C(IC)$ has no cycles and *RIC-cyclic* otherwise. □

Example 2.3 (example 2.2 continued) The contracted dependency graph $\mathcal{G}^C(IC)$ is obtained by replacing in $\mathcal{G}(IC)$ the edges 1 and 2 and their end vertices by a vertex labelled with $\{Q, R, S\}$. The contracted dependency graph, shown in Figure 2.2(a), has no loops, therefore IC is RIC-acyclic. For $IC' = IC \cup \{T(x, y) \to R(y)\}$, all the vertices in $\mathcal{G}(IC)$ belong to the same connected component. $\mathcal{G}(IC')$ and $\mathcal{G}^C(IC')$ are

---

[4]Notice that for every $c \in \mathcal{C}(\mathcal{G}(IC_U))$, it holds $c \in \mathcal{C}(\mathcal{G}(IC))$.

in Figures 2.2(b) and 2.2(c), respectively. Since there is a self-loop in $\mathcal{G}^C(IC)$, IC' is *not* RIC-acyclic. □

(a) $\mathcal{G}^C(IC)$ (b) $\mathcal{G}(IC')$ (c) $\mathcal{G}^C(IC')$
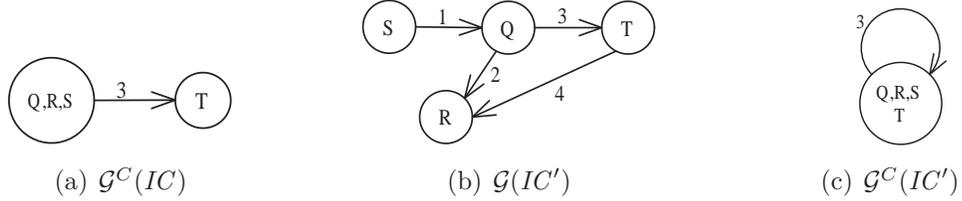
Figure 2.2: Contracted dependency graphs of Example 2.3

As expected, a set of UICs is always RIC-acyclic.

2.2 Repairs and Consistent Query Answering (CQA)

When repairing an inconsistent database with respect to a set IC of integrity constraints by inserting and deleting tuples, we need to be able to determine the distance between the database and the repaired version.

Definition 2.3 [Arenas *et al.*, 1999] The *distance* between two database instances D_1 and D_2 is their symmetric difference $\Delta(D_1, D_2) = (D_1 - D_2) \cup (D_2 - D_1)$. □

We want to have repairs that minimally differ from the original database.

Definition 2.4 [Arenas *et al.*, 1999] Given a database instance D without null values and possibly inconsistent with respect to a set of constraints IC, we say that the instance D' is a *repair* of D with respect to IC iff D' is compatible with Σ, $D' \models IC$, and $\Delta(D, D')$ is minimal under set inclusion in the class of instances that satisfy IC and are compatible with Σ. The set of repairs of D with respect to IC is denoted with $Rep(D, IC)$. □

This definition of repair, assumes that the basic repair operations are tuple deletions and insertions. Alternative semantics have been considered: tuple updates for numerical values [Wijsen, 2003; Wijsen, 2005; Bertossi *et al.*, 2005a; Bertossi *et al.*, 2005c; Bertossi *et al.*, 2005b; Bertossi *et al.*, 2005d; Lopatenko and Bravo, 2006], only tuple deletions [Chomicki and Marcinkowski, 2005b; Chomicki and Marcinkowski, 2002], and priority of tuple insertions over deletions [Lembo *et al.*, 2002].

Example 2.4 Consider the relational schema $Book(author, name, publYear)$, a data-base $D = \{Book(kafka, metamorph, 1915), Book(kafka, metamorph, 1919)\}$ and the functional dependency $FD : author, name \rightarrow publYear$, that can be expressed using form (2.1) as $\forall xyzw(Book(x, y, z) \wedge Book(x, y, w) \rightarrow z = w)$. Instance D is inconsistent with respect to FD, and has two repairs: $D_1 = \{Book(kafka, metamorph, 1915)\}$ and $D_2 = \{Book(kafka, metamorph, 1919)\}$. The respective distances are $\Delta(D, D_1) = \{Book(kafka, metamorph, 1919)\}$ and $\Delta(D, D_2) = \{Book(kafka, metamorph, 1915)\}$. The instance $D_3 = \emptyset$ is not a repair since $\Delta(D, D_3) = \{Book(kafka, metamorph, 1915), Book(kafka, metamorph, 1919)\} \not\supsetneq \Delta(D, D_1)$. $\square$

The definition of repair given in [Arenas *et al.*, 1999] implicitly ignores the possible presence of null values. Similarly, in [Arenas *et al.*, 2003; Barceló; and Bertossi, 2003; Calì *et al.*, 2003a], that followed the repair semantics in [Arenas *et al.*, 1999], no null values were considered in databases or repairs.

Example 2.5 Consider the database D below and the RIC: $Reg(ID, Code) \rightarrow \exists Name\ Student(ID, Name)$.

$D:$ Reg	$\underline{ID}$	$\underline{Code}$		$Student$	$\underline{ID}$	$Name$
	21	$C15$			21	Ann
	34	$C18$			45	$Paul$

D is inconsistent, because there is no tuple in $Student$ for tuple $Reg(34, C18)$. The

database can be minimally repaired by deleting the inconsistent tuple or by inserting a new tuple into table *Student*. In the latter case, since the value for attribute *Name* is unknown, we should consider repairs with all the possible values in the domain. Therefore, for the repair semantics in [Arenas *et al.*, 1999], the repairs are of the following two forms:

Reg	ID	Code
	21	C15

Student	ID	Name
	21	Ann
	45	Paul

Reg	ID	Code
	21	C15
	34	C18

Student	ID	Name
	21	Ann
	45	Paul
	34	μ

for all the possible values of μ in the domain, obtaining, possibly, infinite repairs. □

Definition 2.5 [Arenas *et al.*, 1999] Given a database D, a set of ICs IC, and a query $Q(\bar{x})$, a ground tuple $\bar{t}$ is a *consistent answer* to Q with respect to IC in D iff for every $D' \in Rep(D, IC)$, $D' \models Q[\bar{t}]$. If Q is a sentence (boolean query), then *yes* is a consistent answer iff $D' \models Q$ for every $D' \in Rep(D, IC)$. Otherwise, the consistent answer is *no*. □

Example 2.6 (example 2.4 continued) The query Q_1 : *Book(kafka, metamorph, 1915)* does not have *Yes* as a consistent answer, because it is not true in every repair. Query $Q_2(y)$: $\exists x \exists z Book(x, y, z)$ has $y = metamorph$ as a consistent answer. Query $Q_3(x)$: $\exists z Book(x, metamorph, z)$ has $x = kafka$ as a consistent answer. □

The problem of deciding if a tuple is a consistent answer to a query with respect to a set of universal and referential ICs is undecidable for this repair semantics [Calì *et al.*, 2003a].

2.3   Disjunctive Logic Programs and Stable Models Semantics

The stable models semantics was introduced in [Gelfond and Lifschitz, 1988] to give a semantics to logic programs with weak negation. It was later extended to disjunctive logic programs [Gelfond and Lifschitz, 1991; Przymusinski, 1991]. By now it is the standard semantics for such programs.

A disjunctive logic program Π is a finite set of rules r, of the form :

$$A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n, \ not \ C_1, \ldots \ not \ C_m.$$

where $A_1, \ldots, A_k, B_1, \ldots, B_n, C_1, \ldots, C_m$ are atoms. The disjunction $A_1 \vee \cdots \vee A_n$ is the head of rule r, while the conjunction $B_1, \ldots, B_m, \ not \ C_1, \ldots \ not \ C_k$ is the body of r. A rule with $k = 0$ is a program denial constraint. A rule with $m = n = k = 0$ is called a *fact*, and we will omit the arrow in this case. Let $rule(\Pi)$ be the set of rules in a program Π. A program is called *normal* if $k = 1$ for every rule. A program is *positive* if $m = 0$ for every rule.

For any program Π, let U_Π (the Herbrand Universe) [Lloyd, 1987] be the set of all constants appearing in Π. In case no constant appears in P, an arbitrary constant is added to U_Π. Let the Herbrand Base [Lloyd, 1987], B_Π, be the set of all ground atoms constructible from the predicate symbols (except for built-in predicates) appearing in Π , and the constants of U_Π. The ground instantiation of a rule r, $Ground(r)$, denotes the set of rules obtained by applying all possible substitutions from the variables in r to elements of U_Π. For any program Π, $Ground(\Pi) = \bigcup_{r \in rules(\Pi)} Ground(r)$.

An interpretation I for Π is a set of ground atoms, i.e., $I \subseteq B_\Pi$. A positive ground rule $A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n$ is true in an interpretation I if $\{B_1, \ldots, B_n\} \subseteq I$ and $(\cup_{i=1}^k A_i) \cap I \neq \emptyset$ or if $\{B_1, \ldots, B_n\} \not\subseteq I$. A interpretation I is an model of a program Π if all the rules in it are true in I. A model $\mathcal{M}$ of Π is minimal iff there is no other

model $\mathcal{M}'$ of Π such that $\mathcal{M}' \subsetneq \mathcal{M}$.

A model $\mathcal{M}$ of a disjunctive program Π is *stable* iff $\mathcal{M}$ is a minimal model of $Ground(\Pi)^{\mathcal{M}}$, where the positive ground program $Ground(\Pi)^{\mathcal{M}}$ is defined as $\{A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n \mid (A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n, \; not \; C_1, \; \ldots, \; not \; C_m) \in rule(Ground(\Pi))$ and, for every $1 \leq i \leq m$, $\mathcal{M} \not\models C_i\}$ [Gelfond and Lifschitz, 1991; Przymusinski, 1991].

Since a program Π can have multiple stable models, we can consider two possible semantics to answer queries. Given a query $Q(\bar{x})$, a ground tuple $\bar{t}$ is an answer to Q in Π under the *skeptical semantics* (*brave semantics*) if $Q[\bar{t}]$ is true in all (at least one) stable model.

Chapter 3

Thesis Objectives

There are many relevant and promising open problems in the area of consistent query answering in inconsistent databases. The research in this thesis is focused on extending the current methodologies for retrieving consistent information from databases to a wider class of constraints, exploring alternative semantics, and applying the obtained results to other areas such as data integration systems and peer data management systems.

1. Null values and consistency:

   (a) Define a semantics for satisfaction of constraints in databases with null values, that extends and unifies the semantics currently used in commercial DBMS that follow the SQL Standard, such as IBM DB2.

   (b) Extend the semantics for satisfaction of constraints to a query answering semantics.

2. Consistent Query Answering (CQA) in Relational Databases:

   (a) Extend current methodologies [Barceló and Bertossi, 2002; Barceló; and Bertossi, 2003] that use repair logic programs to deal with referential integrity constraints (RICs). Analyze possible ways of optimizing the repair logic program.

   (b) Consider alternative semantics, such as repairs obtained only by tuple deletions or where insertions have priority over deletions.

(c) Complexity Analysis of CQA for the different semantics. Classes of lower complexity are also identified.

3. CQA in Data Integration Systems (DIS):

   (a) Give a logical specification of a DIS for *open* sources with *local-as-view (LAV)* mappings between sources and the global schema. Explore possible modifications to the specification if *closed* and *exact* mappings are to be considered. Analyze how and when this specification can be used to retrieve certain answers.

   (b) Modify the repair logic program obtained in 1.(a) to work together with the DIS specification. Analyze how and when this specification can be used to retrieve consistent answers.

   (c) Explore modifications of the specification to consider *global-as-view (GAV)* mappings.

4. Peer Data Management Systems (PDM):

   (a) Provide a semantics for query answering from a PDM with trust relationships and local integrity constraints, assuming that only direct neighbors influence the answers of a peer.

   (b) Extend the results in 4.(a) to the transitive case in which neighbors of neighbors can also influence the answers of a peer.

   (c) Modify the repair program of 1.(a) so that it can be used to obtain answers from a peer.

Chapter 4

Null Values and Consistency

This research has its origin in *consistent query answering* (CQA) [Arenas *et al.*, 1999; Bertossi and Chomicki, 2003; Bertossi, 2006]. This is the problem of characterizing and retrieving semantically correct answers to a query from a relational database that may fail to satisfy a given set of integrity constraints (ICs).

Different forms of repair have been studied in the literature to characterize consistent answers. Some of them restore consistency with respect to referential ICs by introducing a single null value [Arenas *et al.*, 2003; Barceló; and Bertossi, 2003; Barceló *et al.*, 2003; Bravo and Bertossi, 2004], and others, labeled (or symbolically different) null values [Calì *et al.*, 2003a]. However, several issues that are relevant to the possible application and use of the notion of CQA in real databases, as implemented in and managed by commercial systems that (possibly only partially) follow the SQL standard [International Organization for Standardization, 2003], have been basically ignored: (a) Possibly several occurrences of a unique unlabeled null value, *null*, can be found already in the database, even before considering repairs. (b) In these systems and in the SQL standard, the notion of satisfaction of ICs in the presence of *null* does not follow a clear or general logic. (c) The "logics" for IC satisfaction and query answering are not uniformly integrated.

It is clear that, in order to define and do CQA in real DBMSs that use null values in practice, it is necessary to address these issues first. These problems are also of theoretical and practical relevance in themselves, independently from CQA.

In this chapter we provide: (a) A precise, uniform, and logic-based definition of IC satisfaction in SQL databases; and (b) a notion of logical satisfaction as applied to query answering that smoothly extends the same notion for ICs. As a result of this research we are able to propose a clear and precise formalization in first-order logic of the notions of integrity satisfaction in databases that conform to the SQL standard SQL:2003 [International Organization for Standardization, 2003], that is implemented in commercial DBMSs, like IBM DB2 UDB. In particular, we propose a logical reconstruction of the way null values are usually treated in commercial DBMSs.

In order to give a precise semantics for integrity constraint satisfaction in the presence of *null*, we introduce the notion of *relevance of an attribute for the occurrence of null in a relation.* This is because the position *null* in a relation is crucial to check the satisfaction of ICs.

In this chapter, we do not consider database repairs or CQA. These topics will be addressed in Chapter 5. There, we will not only consider databases with *null*, but we will use *null* to restore consistency of referential constraints.

There is no agreement in the literature on the semantics of null values in relational databases. The same applies to *incomplete databases* (null values can be seen as a way of representing incomplete information). There are several different proposals in the research literature [Reiter, 1984; Atzeni and Morfuni, 1986; Levene and Loizou, 1997a; Lien, 1982] (see [van der Meyden, 1998] for a survey), in the SQL standard [Türker and Gertz, 2001; International Organization for Standardization, 2003], and also implicit semantics through the way null values are handled in commercial DBMSs.

In general, the literature on incomplete databases and null values in databases deals with several, possibly symbolically different, null values (labeled nulls), or with different *kinds* of null values that capture different intuitions behind them, e.g. nulls

that represent information that is withheld, inapplicable, uncertain, missing, unknown, etc. Different null constants can be used for each of these different interpretations [Libkin, 1995]. Other publications on incomplete databases consider that a null value represents a collection of possible values of the database domain, so that in this way we are dealing not with a single database, but with a set of *possible worlds* [Imielinski and Lipski, 1984; Grahne, 1991]. In [Reiter, 1984], a null value is considered a value taken from a set of special constants to which the *unique names assumption* does not apply, i.e., different constants may denote the same object. In consequence, we can have several symbolically different null values in a database, but they could be interpreted in the same way.

In spite of all the existing semantics that have been proposed for null values, the logical foundations of databases with null values as they are found in the practice of databases and managed by commercial DBMSs has not received much attention [Levene and Loizou, 1999b]. The implemented "semantics" of null values as specified in the SQL standard and implemented in commercial DBMSs has been criticized in the database literature [van der Meyden, 1998].

The implemented semantics for query answering follows some sort of three-valued logic. However, it has been noted that under this three-valued logic, some tuples will not be returned as answers to some tautological queries [Grant, 1977; Codd, 1979]. For example if we ask for all the employees with $salary > 3000$ or $salary \leq 3000$, any employee with $salary = null$ will be left out of the answer set. Date and Warden (1990) show that the EXIST operator has some unexpected behavior as a consequence of how operators behave in the presence of *null*. For example, the union[1] of two relations where each contains tuple $(a, null)$ would be expected to return $\{(a, null), (a, null)\}$ since the null value in each tuple might be different, but in the

---

[1]Even tough most of the operations in DBMSs use the bag semantics, the operations union and intersection return by default the set semantics.

SQL standard a relation with only one tuple (a, null) is returned [van der Meyden, 1998].

As mentioned before, logically cleaner versions of null value semantics have been proposed by the research community, but they have not been adopted by commercial systems in spite of the criticisms they have received.

In this chapter, we start from the fact that we have working commercial implementations with a specific semantics. In many aspects, they still lack a clear logic, and for this reason, it is better to have a clear picture of the logics behind their constructs and processes. From this point of view, our research sheds some light on this scenario, by providing a *logical reconstruction* that can be uniformly applied to a wide class of ICs, in particular to those that are used in common database applications, but going beyond the type of ICs supported by commercial DBMS.

Starting from the semantics of IC satisfaction given in Section 4.1, we proceed to develop in Section 4.2 a semantics for query answering in databases with *null*, which amounts to developing a logical notion of formula satisfaction in databases conceived as first-order structures. Since such a notion should be applicable to satisfaction of sentences, it should also be applicable to satisfaction of ICs. We show that IC satisfaction from this perspective is compatible with the previously defined notion of IC satisfaction.

In this paper we consider a *set semantics* for databases, as apposed to a *bag semantics* that would accept repeated tuples in a databases. An extension of our semantics for IC satisfaction and query answering in order to properly deal with bags is left for future work.

4.1 IC Satisfaction in Databases with Null Values

Not even within the SQL standard there is a homogenous and global semantics of integrity constraint satisfaction in databases with null values; rather, different definitions of satisfaction are given for each type of constraint. Actually, in the case of foreign key constraints, three different semantics are suggested (*simple-*, *partial-* and *full-match*)[International Organization for Standardization, 2003]. Commercial DBMSs implement only the simple-match semantics for foreign key constraints [IBM, 2006; Oracle, 2005; MySQL, 2006; Microsoft, 2006; PostgreSQL, 2006; Sybase, 2006]. Furthermore the "position" where *null* appears in a database or in the IC is relevant for checking IC satisfaction.

In [Barceló *et al.*, 2003; Bravo and Bertossi, 2004], in the context of CQA, a semantics for null values was adopted, according to which a tuple with a null value in any of its attributes would not be the cause for any inconsistencies. In other words, it would not be necessary to check tuples with null values with respect to possible violations of ICs (except for *NOT NULL-* constraints, of course). This assumption is consistent in some cases with the practice of DBMSs, e.g. in IBM DB2 UDB. Here, we will propose a semantics that is less liberal in relation to the participation of null values in inconsistencies; a sort of compromise solution between the different alternatives available, that coincides with the portion of the SQL standard that is implemented in DBMSs. This is, we will extend the simple-match for foreign keys and we will not consider assertion constraints [International Organization for Standardization, 2003] which are not currently implemented in commercial DBMSs [IBM, 2006; Oracle, 2005; MySQL, 2006; Microsoft, 2006; PostgreSQL, 2006; Sybase, 2006].

We need some examples first, to motivate our approach.

Example 4.1 For a set of IC containing only of $\psi_1 : \forall xyz(P(x, y, z) \rightarrow R(y, z))$, the

database $D = \{P(a, b, null)\}$ would be: (a) Consistent with respect to the semantics in [Barceló *et al.*, 2003; Bravo and Bertossi, 2004], because *null* is in the tuple. (b) Consistent with respect to the simple-match semantics of SQL:2003 [International Organization for Standardization, 2003], because *null* is in one of the attributes in the set $\{P[2], P[3], R[1], R[2]\}$ of attributes that are relevant to check the constraint. (c) Inconsistent with respect to the partial-match semantics in SQL:2003, because there is no tuple in R with a value b in its first attribute. (d) Inconsistent with respect to the full-match semantics in SQL:2003, because *null* cannot be in an attribute that is referencing a different table.

If we consider, instead of ψ_1, the constraint $\psi_2 \colon \forall xyz(P(x, y, z) \rightarrow R(x, y))$, the same database would be consistent only for the semantics in [Barceló *et al.*, 2003; Bravo and Bertossi, 2004]. □

Example 4.2 Consider a database with a table *Course* that stores courses with the professors who teach them, along with the terms in which they are taught; and a table *Exp* that stores each professor in each course with the number of times (s)he has taught the course. We have a foreign key constraint based on the RIC $\forall xyz(Course(x, y, z) \rightarrow \exists w\ Exp(y,\ x,\ w))$, together with the constraint expressing that table *Exp* has $\{ID, Code\}$ as a primary key. In commercial DBMSs, primary keys cannot contain null values.

Now consider instance D_1:

Course	Code	ID	Term	Exp	Code	ID	Times
	CS27	21	W04		CS27	21	3
	CS18	34	null		CS18	34	null
	CS50	null	W05		CS32	45	2

In IBM DB2, this database is accepted as consistent. The null values in columns *Term* and *Times* are not relevant to check the satisfaction of the constraints. In order to

check the constraint, the only attributes that we need to pay attention to are *ID* and *Code*. If *null* is in one of these attributes of table *Course*, the tuple is considered to be consistent, without checking table *Exp*. For example, *Course(CS50, null, W05)* has a null value in *ID*, therefore DB2 does not check if there is a tuple in *Exp* that satisfies the constraint. It does not even check that there exists a tuple in *Exp* with attribute *Code = CS50*.

This behavior for foreign key constraints is called "simple-match" in the SQL standard, and is the one implemented in commercial DBMS such as IBM DB2 [IBM, 2006], Oracle [Oracle, 2005], MySQL [MySQL, 2006], Microsoft SQL Server [Microsoft, 2006], PostgreSQL [PostgreSQL, 2006] and Sybase ACE [Sybase, 2006]. The "partial-match" and "full-match" would not accept the database as consistent, because partial-match would require *Exp* to have a tuple of the form (any non-null value, *CS50*, any value); and full-match would not allow a tuple with *null* in attributes *ID* or *Code* in table *Course*.

If we try to insert tuple (*CS41*, 18, *null*) into table *Course*, it would be rejected by DB2. This is because the attributes *ID* and *Code* are relevant to check the constraint and are different from *null*, but there is no tuple in *Exp* with *ID* = 18 and *Code = CS41*. □

Example 4.3 Consider a database with tables *Person* and *Phone*, where *Number* is the primary key of *Phone*, and there is a foreign key from attribute *Phone* in *Person* to table *Phone*, i.e., $\forall xyz(Person(x, y, z) \rightarrow \exists w Phone(z, w))$. The following database instance is accepted as a consistent state in DB2.

Person	*ID*	*Name*	*Phone*
	182	*Ann*	*null*

Phone	*Number*	*Provider*

This is an example where the *null* can be interpreted as a non-existent value, that is, *Ann* might *not* have a phone number and, therefore it is not a problem to have no

tuples in table *Phone*. Here, the relevant attributes needed to check the satisfaction of the foreign key are *Phone, ID*. □

Example 4.4 Consider the check constraint $\forall xyz(Emp\ (x,y,z) \rightarrow z > 100)$ and the database D:

Emp	ID	Name	Salary
	32	*null*	1000
	41	*Paul*	*null*

DB2 accepts this database instance as consistent. Here, in order to check the satisfaction of the constraint, we only need to verify that the attribute *Salary* is greater than 100; therefore the only attribute that is relevant to check the constraint is *Salary*. DBMSs will accept as consistent any state where the condition (the consequent) evaluates to *true* or *unknown*. The latter is the case here. Tuple (32, *null*, 50) could not be inserted, because $Salary > 100$ evaluates to *false*. Notice that the null values in attributes other that *Salary* are not even considered in the verification of the satisfaction. □

DBMSs use a bag semantics instead of the set semantics, that is, a table can have two copies of the same tuple. This raises some issues when checking the satisfaction of primary keys.

Example 4.5 Since the SQL standard allows duplicate rows, i.e., uses bag semantics, it is possible to have the database D:

P	A	B
	a	b
	a	b

If this database had A as the primary key, then D would not have been accepted as a consistent state, i.e., the insertion of the second tuple $P(a, b)$ would have been rejected. This is one of the cases in which the SQL standard deviates from the relational model, where duplicates of a row are not considered. In a commercial DBMS a primary key is checked by adding an index to the primary key and then ensuring that there are no duplicates. Therefore, if we try to check the primary key by using the associated functional dependency $\forall xyz(P(x, y) \wedge P(x, z) \rightarrow y = z)$, we would not have the same semantics. This is because D satisfies the functional dependency in this classical, first-order representation. $\square$

With the type of first-order constraints that we are considering, we cannot enforce a bag semantics, therefore we will assume that D has no duplicate tuples.

In order to develop a null-value semantics that goes beyond the ICs supported by DBMSs, we analyze other examples.

Example 4.6 Consider the UIC $\forall xyzstuw(Person(x, y, z, w) \wedge Person(z, s, t, u) \rightarrow u > w + 15)$, and the database D:

Person	Name	Dad	Mom	Age
	Lee	Rod	Mary	27
	Rod	Joe	Tess	55
	Mary	Adam	Ann	null

This constraint can be considered as an extension of check constraints, that, instead of checking a condition for one row of the table, it considers multiple rows (a multi-row check constraint). Therefore, we can naturally extend the semantics for single-row check constraints, by taking D as consistent iff the condition evaluates to *true* or *unknown*. In this case, D would be consistent, because the condition evaluates to

unknown for $u = null$ and $w = 27$, and to *true* in the other cases. Here, the relevant attributes to check the IC are *Name, Mom, Age*. □

The following example shows that it is not clear how to extend the semantics of satisfaction of constraints in DBMSs to inclusion dependencies that are not foreign key constraints.

Example 4.7 Consider the UIC $\forall xyz(Course(x, y, z) \rightarrow Employee(y, z))$ and the database D:

Course	Code	Term	ID
	CS18	W04	34

Employee	Term	ID
	W04	null

Since $\{Term, ID\}$ is not a primary key of *Employee*, the constraint is not a foreign key constraint, and therefore it is not supported by DBMS [IBM, 2006; Oracle, 2005; MySQL, 2006; Microsoft, 2006; PostgreSQL, 2006; Sybase, 2006]. Also, in contrast to foreign key constraints, now we can have *null* in the referenced attributes. There is no class of constraints used in commercial DBMSs that we can use as a hint on how we can extend the satisfaction semantics to this case. □

In order to decide what semantics to give for constraints like the one in Example 4.7, we appeal to the literature. In [Levene and Loizou, 1997a], the satisfaction of this type of constraints is defined as follows: An IND $P[\bar{X}] \subseteq Q[Y]$ is satisfied if, for every tuple $t_1 \in P$, there exists a tuple $t_2 \in Q$, such that $t_1[\bar{X}]$ provides *less or equal information* than $t_2[\bar{Y}]$. The concept of less or equal information is defined as follows:

Definition 4.1 [Levene and Loizou, 1997a] (a) A constant c provides less or equal information than a constant d, denoted $c \sqsubseteq d$, iff c is null or $c = d$. (b) A tuple $t_1 = (c_1, \ldots, c_n)$ provide less or equal information than $t_2 = (d_1, \ldots, d_n)$, denoted $t_1 \sqsubseteq t_2$, iff $c_i \sqsubseteq d_i$ for every $i = 1, \ldots, n$. (c) $t_1 \sqsubset t_2$ iff $t_1 \sqsubseteq t_2$ and $t_1 \neq t_2$. (d) An

IND $P[\bar{X}] \subseteq Q[Y]$ is satisfied if, for every tuple $t_1 \in P$, there exists a tuple $t_2 \in Q$ such that $t_1[\bar{X}] \sqsubseteq t_2[\bar{Y}]$.   □

Example 4.8 (example 4.7 continued) For the tuple *Course*(*CS18,W04,34*) there is no tuple in table *Employee* that provides less or equal information. For example, (*W04,34*) $\not\sqsubseteq$ (*W04,null*). Therefore, we consider the database to be inconsistent with respect to the constraint. Note that the only attributes that are relevant to check the constraint are *Term* and *ID*.   □

Examples 4.2, 4.3, 4.4, 4.6 and 4.8 show that there are some attributes that are "relevant" when the satisfaction of a constraint is checked against a database.

Definition 4.2 For t a term, i.e., a variable or a domain constant, let $pos^R(\psi, t)$ be the set of positions in predicate $R \in \mathcal{R}$ where t appears in ψ. The set $\mathcal{V}$ of *relevant variables* for an IC ψ of the form (2.1) is $\mathcal{V}(\psi) = \{x \mid x$ is a repeated variable in $\psi\}$. The set $\mathcal{A}$ of *relevant attributes* for ψ is

$\mathcal{A}(\psi) = \{R[i] \mid x \in \mathcal{V}(\psi)$ and $i \in pos^R(\psi, x)\} \cup$

$\qquad \{R[i] \mid c$ is a constant in ψ and $i \in pos^R(\psi, c)\}.^2$ □

For an IC of the form (2.1), all relevant variables have at least one occurrence in the antecedent of ψ, are variables in joins or appear in φ.

If a built-in predicate in φ has a redundant or trivial occurrence of a variable, e.g. $x = x$ for x appearing in a database atom, then this would have the effect of transforming an attribute in relevant when it does not need to be. For example, the constraint $\forall xy(T(x, y) \rightarrow x \leq 5 \vee y \geq y)$ is equivalent to $\forall xy(T(x, y) \rightarrow x \leq 5)$, but the latter has relevant attributes x and y and the former only x. The specifier of ICs should be aware of this issue. Furthermore, we should observe that an apparently tautological occurrence of $x = x$ is not such, because this equality has a different

---

[2]Remember that $R[i]$ denotes a position in relation R.

meaning than the classical when applied to null values. As a consequence, it is reasonable to consider this variable or attribute as relevant (with respect to null values).

Definition 4.3 For a set of attributes $\mathcal{A}$ and a predicate $P \in \mathcal{R}$, we denote by $P^{\mathcal{A}}$ the predicate P restricted to (or projected onto) the attributes in $\mathcal{A}$. $D^{\mathcal{A}}$ denotes the database D with all its database atoms projected onto the attributes in $\mathcal{A}$, i.e., $D^{\mathcal{A}} = \{P^{\mathcal{A}}(\Pi_{\mathcal{A}}(\bar{t})) \mid P(\bar{t}) \in D\}$, where $\Pi_{\mathcal{A}}(\bar{t})$ is the projection on $\mathcal{A}$ of tuple $\bar{t}$. $D^{\mathcal{A}}$ has the same underlying domain $\mathcal{U}$ as D. □

Example 4.9 Consider a UIC $\psi : \forall xyz(P(x,y,z) \rightarrow R(x,y))$, and D:

P	A	B	C
	a	5	a
	b	3	a

R	A	B
	a	5
	a	2

Since x and y appear twice in ψ, $\mathcal{A}(\psi) = \{P[1], R[1], P[2], R[2]\}$. The value in z should not be relevant to check the satisfaction of the constraint, because we only want to make sure that the values in the first two attributes in P also appear in R. This, checking its satisfaction is equivalent to checking if $\forall xy(P^{\mathcal{A}(\psi)}(x,y) \rightarrow R^{\mathcal{A}(\psi)}(x,y))$ is satisfied by $D^{\mathcal{A}(\psi)}$. For a more complex constraint, such as $\gamma : \forall xyzw(P(x,y,z) \wedge R(z,w) \rightarrow \exists v R(x,v) \vee w > 3)$, variable x is relevant to check the implication, z is needed to do the join, and w is needed to check the comparison, therefore $\mathcal{A}(\gamma) = \{P[1], R[1], P[3], R[2]\}$.

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

$P^{\mathcal{A}(\psi)}$	A	B
	a	5
	b	3

$R^{\mathcal{A}(\psi)}$	A	B
	a	5
	a	2

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

$P^{\mathcal{A}(\gamma)}$	A	C
	a	a
	b	a

$R^{\mathcal{A}(\gamma)}$	A	B
	a	5
	a	2

□

An important observation we can make from Examples 4.2, 4.4, 4.6 and 4.7 is that, roughly speaking, a constraint is satisfied if any of the relevant attributes has *null* or the constraint is satisfied in the traditional way (i.e. first-order satisfaction and *null* treated as any other constant).

Definition 4.4 A constraint ψ of the form (2.1):

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

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

$$\forall \bar{x}(\bigwedge_{i=1}^{m} P_i^{\mathcal{A}(\psi)}(\bar{x}_i) \rightarrow (\bigvee_{v_j \in \mathcal{V}(\psi)} v_j = null \vee \bigvee_{j=1}^{n} Q_j^{\mathcal{A}(\varphi)}(\bar{y}_j) \vee \varphi)), \qquad (4.1)$$

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

We can see from Definition 4.4 that there are basically two cases for constraint satisfaction: (a) If *null* is in any of the relevant attributes in the antecedent, then the constraint is satisfied. (b) If *null* does not appear in the relevant attributes, then the second disjunct in the consequent of formula (4.1) has to be checked, i.e., the consequent of the IC restricted to the relevant attributes. This can be done as usual, treating *null* as any other constant.

Formula (4.1) is a direct reduction of Formula (2.1). In particular, if the constraint is universal, so is the transformed version. Notice that Formula (4.1) is domain independent, and therefore its satisfaction can be checked on the active domain.

As mentioned before, the semantics for IC satisfaction introduced in [Bravo and Bertossi, 2004] considers that tuples with *null* never generate any inconsistency, even when *null* is not in a relevant attribute. For example, under the semantics in [Bravo and Bertossi, 2004], the instance $\{P(b, null)\}$ would be consistent with respect to the IC $\forall xy(P(x,y) \rightarrow R(x))$, but intuitively the constraint implies that every element in the first attribute of table P should be in the first attribute of table R, and this is not the case for $\{P(b, null)\}$. The new semantics corrects this, and adjusts itself to the semantics implemented in commercial DBMS.

Example 4.10 Given the ICs: (a) $\forall xyz(P(x,y,z) \rightarrow R(x,y))$, (b) $\forall x(T(x) \rightarrow \exists yz P(x, y, z))$, the database instance D below is consistent.

P	A	B	C
	a	d	e
	b	*null*	g

R	D	E
	a	d

T	F
	b

For (a), the relevant variables to check the constraint are $\mathcal{V}_1 = \{x, y\}$, therefore $\mathcal{A}_1 = \{P[1], R[1], P[2], R[2]\}$; and for (b), $\mathcal{V}_2 = \{x\}$; therefore $\mathcal{A}_2 = \{P[1], T[1]\}$.

$D^{\mathcal{A}_1}$:

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

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

$D^{\mathcal{A}_2}$:

$P^{\mathcal{A}_2}$	A
	a
	b

$T^{\mathcal{A}_2}$	F
	b

To check if $D \models_N \forall xyz(P(x,y,z) \rightarrow R(x,y))$, we need to check if $D^{\mathcal{A}_1} \models \forall xy(P^{\mathcal{A}_1}(x,y) \rightarrow (x = null \lor y = null \lor R^{\mathcal{A}_1}(x,y)))$. For $x = a$ and $y = d$, $D^{\mathcal{A}_1} \models P^{\mathcal{A}_1}(a,d)$, but

none of them is *null*, therefore we need to check if $D^{\mathcal{A}_1} \models R^{\mathcal{A}_1}(a, d)$. This is true, therefore the constraint is satisfied for $x = a$ and $y = d$. Now, for $x = b$ and $y = null$, $D^{\mathcal{A}_1} \models P^{\mathcal{A}_1}(b, null)$, and since $y = null$, the constraint is satisfied. The same analysis can be done to prove that D satisfies constraint (b), i.e., by checking $D^{\mathcal{A}_2} \models \forall x (T^{\mathcal{A}_2}(x) \rightarrow (x = null \vee P^{\mathcal{A}_2}(x)))$.

If we add tuple $P(f, d, null)$ to D, it would become inconsistent with respect to constraint (a), because $D^{\mathcal{A}_1} \not\models (P^{\mathcal{A}_1}(f, d) \rightarrow (f = null \vee d = null \vee R^{\mathcal{A}_1}(f, d)))$. $\square$

Example 4.11 Consider the IC ψ: $\forall xywz((P_1(x, y, w) \wedge P_2(y, z)) \rightarrow \exists u \, Q(x, z, u))$ and the database D:

P_1	A	B	C
	a	b	c
	d	$null$	c
	b	e	$null$
	$null$	b	b

P_2	D	E
	b	a
	e	c
	d	$null$
	$null$	b

Q	F	G	H
	a	a	c
	b	$null$	c
	b	c	d
	$null$	c	a

Variables x, y and z are relevant to check the constraint, therefore the set of relevant attributes is $\mathcal{A}(\psi) = \{P_1[1], P_1[2], P_2[1], P_2[2], Q[1], Q[2]\}$. Then we need to check if $D^{\mathcal{A}(\psi)} \models \forall xyz((P_1^{\mathcal{A}(\psi)}(x, y) \wedge P_2^{\mathcal{A}(\psi)}(y, z)) \rightarrow (x = null \vee y = null \vee z = null \vee Q^{\mathcal{A}(\psi)}(x, z)))$, where $D^{\mathcal{A}(\psi)}$ is

$P_1^{\mathcal{A}(\psi)}$	A	B
	a	b
	d	$null$
	b	e
	$null$	b

$P_2^{\mathcal{A}(\psi)}$	D	E
	b	a
	e	c
	d	$null$
	$null$	b

$Q^{\mathcal{A}(\psi)}$	F	G
	a	a
	b	$null$
	b	c
	$null$	c

When checking the satisfaction of $D^{\mathcal{A}(\psi)} \models \psi^N$, *null* is treated as any other constant. For example for $x = d$, $y = null$ and $z = b$, the antecedent of the rule is satisfied since

$P_1^{\mathcal{A}(\psi)}(d, null) \in D^{\mathcal{A}}$ and $P_2^{\mathcal{A}(\psi)}(null, b) \in D^{\mathcal{A}}$. If $null$ had been treated as a special constant, with no unique names assumption applied to it, the antecedent would have been false. For these values the consequent is also satisfied, because $y = null$ is true. In this example, $D^{\mathcal{A}(\psi)} \models \psi^N$, and the database satisfies the constraint. □

Notice that in a database without $null$, Definition 4.4 coincides with the traditional, first-order definition of IC satisfaction.

Example 4.12 (example 4.9 continued) In order to check if $D \models_N \psi$, we need to check if $D^{\mathcal{A}(\psi)} \models \psi^N$, with $\psi^N : \forall xyz (P^{\mathcal{A}(\psi)}(x, y) \rightarrow (x = null \vee y = null \vee R^{\mathcal{A}(\psi)}(x, y)))$. It is easy to see that, since D has no $null$, checking $D^{\mathcal{A}(\psi)} \models \psi^N$ is equivalent to checking $D \models \psi$. □

Our semantics is a natural extension of the semantics used in commercial DBMSs. Note that: (a) in a DBMS there will never be a join between $null$ and another value ($null$ or not); (b) Any check constraint with comparison, e.g $<, >, =$, will never create an inconsistency when comparing $null$ with any other value. These two features justify our decision in Definition 4.4 to include the attributes in the joins and the elements in φ among the attributes that are checked to be $null$, because if there is $null$ in them, an inconsistency will never arise.

4.1.1 The *IsNull* Predicate

There is a very important constraint widely used in DBMSs that we have not dealt with yet: the NOT NULL constraint (NNC). This constraint prevents certain attributes from taking the value $null$.

NNCs are of particular interest for attributes in primary key constraint. In fact, all the attributes in a primary key have to be set to NOT NULL [IBM, 2006; Oracle, 2005;

MySQL, 2006; Microsoft, 2006; PostgreSQL, 2006; Sybase, 2006]. A *unique constraint* has the same characteristics as a primary key, but without the NOT NULL constraint.

To express a NNC, we introduce a special predicate $IsNull(\cdot)$, with $IsNull(c)$ true iff c is *null*, instead of using the built-in comparison atom $c = null$, because in traditional DBMS this equality would be always evaluated as *unknown* (as observed in [Reiter, 1984], the *unique names assumption* does not apply to *null*).

Definition 4.5 A *NOT NULL*-constraint (NNC) is a denial constraint of the form

$$\bar{\forall}\bar{x}(P(\bar{x}) \wedge IsNull(x_i) \rightarrow \textbf{false}), \tag{4.2}$$

where $x_i \in \bar{x}$ is in the position of the attribute that cannot take the value *null*. □

Notice that a NNC is not of the form (2.1), because it contains the special predicate *IsNull*.

Definition 4.6 A NNC ψ of the form (4.2), is satisfied by a database D with *null*, denoted $D \models_N \psi$, iff

$$D \models \forall\bar{x}((P(\bar{x}) \wedge x_i = null) \rightarrow \textbf{false}) \tag{4.3}$$

with *null* treated as any other constant. □

Example 4.13 Consider the NNC $\psi : \forall xy(P(x, y) \wedge IsNull(y) \rightarrow \textbf{false})$. This constraint is satisfied if $D \models (\forall xy(P(x, y) \wedge y = null \rightarrow \textbf{false}))$. □

We can modify the general form of ICs (see equation (2.1)) to accommodate also NNCs and other variations of constraints with the *IsNull* predicate.

Definition 4.7 A *general integrity constraint* is a sentence $\psi \in \mathcal{L}(\Sigma)$ of the form :

$$\forall \bar{x}((\bigwedge_{i=1}^{m} P_i(\bar{x}_i) \wedge \bigwedge_{k=1}^{a} IsNull(x_k)) \longrightarrow \bigvee_{l=1}^{b} IsNull(x_l) \vee \exists \bar{z}(\bigvee_{j=1}^{n} Q_j(\bar{y}_j, \bar{z}_j) \vee \varphi)), \quad (4.4)$$

where $P_i, Q_j \in \mathcal{R}$, $\bar{x} = \bigcup_{i=1}^{m} \bar{x}_i$, $x_k \subseteq \bar{x}$ for $k = 1, \ldots, a$, $x_l \subseteq \bar{x}$ for $l = 1, \ldots, b$, $\bar{z} = \bigcup_{j=1}^{n} \bar{z}_j$, $\bar{y}_j \subseteq \bar{x}$, $\bar{x} \cap \bar{z} = \emptyset$, $\bar{z}_i \cap \bar{z}_j = \emptyset$ for $i \neq j$, $\bar{z}_j$ does not have repeated variables for $j = 1, \ldots, n$, and $m \geq 1$. $\qquad\square$

We now need to define which are the relevant variables and attributes for a general integrity constraint. In this case, we will also need to separate the relevant variables that are related to the *IsNull* predicate from those that are not.

Definition 4.8 (a) The set $\mathcal{V}^R$ of *restricted relevant variables* for an IC ψ of the form (4.4) is: $\mathcal{V}^R(\psi) = \{x \mid x$ is a repeated variable in ψ except for the variables in the *IsNull* predicate $\}$. (b) The set of variables in *IsNull* predicates in ψ is denoted by $\mathcal{V}^{IsNull}(\psi)$. (c) The set $\mathcal{V}$ of *relevant variables* of ψ is $\mathcal{V}(\psi) = \mathcal{V}^R(\psi) \cup \mathcal{V}^{IsNull}(\psi)$. (d) The set $\mathcal{A}(\psi)$ of *relevant attributes* of ψ is $\{R[i] \mid x \in \mathcal{V}(\psi)$ and $i \in pos^R(\psi, x)\}$[3]. $\square$

For constraints of the form (2.1), if any of the repeated variables in it is *null*, the constraint is satisfied. This is no longer the case when there is an *IsNull* predicate. In fact, in $\forall x(P(x) \wedge IsNull(x) \rightarrow \textbf{false})$, even though x is repeated, if we assign *null* to x, the constraint is not satisfied. This is why we need to add the distinction between the restricted relevant variables and the relevant variables.

The satisfaction of a *general IC* can be determined as in Definition 4.4, but using relevant attributes and variables as defined in Definition 4.8.

---

[3]As defined in Chapter 2, $R[i]$ denotes the attribute in position i in predicate R.

Definition 4.9 A general IC constraint ψ as in (4.4) is satisfied in the database instance D, denoted $D \models_N \psi$, iff $D^{\mathcal{A}(\psi)} \models \psi^N$, where ψ^N is

$$\forall \bar{x}'((\bigwedge_{i=1}^{m} P_i^{\mathcal{A}(\psi)}(\bar{x}_i') \wedge \bigwedge_{k=1}^{a} x_j = null) \rightarrow (\bigvee_{v_j \in \mathcal{V}^R(\psi)} v_j = null \vee \bigvee_{l=1}^{b} x_l = null \vee$$
$$\bigvee_{j=1}^{n} Q_j^{\mathcal{A}(\psi)}(\bar{y}_j) \vee \varphi)), \tag{4.5}$$

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

It is easy to check that Definition 4.9 is an extension of both the semantics of IC satisfaction for ICs without *IsNull* (see Definition 4.4) and the semantics of NNCs satisfaction (see Definition 4.13).

Example 4.14 (example 4.13 continued) The only repeated variable in the NNC ψ, taking into consideration all the variables except for those in the *IsNull* predicate, is x, therefore $\mathcal{A}(\psi) = \{P[1]\}$. By Definition 4.4, it holds $D \models_N \psi$ if $D \models (\forall xy(P(x,y) \wedge y = null \rightarrow \textbf{false}))$. This coincides with Definition 4.6.

Consider now the general IC $\phi : \forall xy(P(x,y) \wedge IsNull(y) \rightarrow Q(x,3))$. Here, the only repeated variable, taking into consideration all the variables except for those in the *IsNull* predicate, is x, therefore $\mathcal{A}(\psi) = \{P[1], Q[1], Q[2]\}$. The position $Q[2]$ is relevant because there is a constant in this position. $D \models_N \phi$ if $D \models (\forall xy(P(x,y) \wedge y = null \rightarrow Q(x,3)))$. $\square$

By adding NNCs, we are able to represent all the common constraints of commercial DBMS, i.e., primary keys, unique constraints, foreign key constraints, check constraints and *NOT NULL*-constraints.

4.1.2 Primary Keys

If we consider a primary key constraint as a set of functional dependencies, and we apply to it the notion of satisfaction of Definition 4.4 via the corresponding FDs, we will obtain a semantics for satisfaction of primary key constraints that is different from the one implemented in DBMSs.

Example 4.15 Consider $D = \{P(a, b), P(a, null)\}$ with primary key $P[1]$. This data-base is not accepted in DB2 as a consistent state. Alternatively, we could try to define the primary key as the functional dependency: $\psi : \forall xyz(P(x, y) \wedge P(x, z) \rightarrow y = z)$. By Definition 4.4, $D \models_N \psi$ iff $D \models (\forall xyz(P(x, y) \wedge P(x, z) \rightarrow x = null \vee y = null \vee z = null \vee y = z))$. Since this is true, the constraint is satisfied for this semantics. Primary keys impose a stronger requirement over the database than their functional dependency versions. $\square$

If we want our semantics to coincide with the portion of the SQL standard implemented in DBMSs, we would need to define a primary key in the following way:

Definition 4.10 Given a predicate $R(x_1, \ldots, x_n)$ and its primary key $\{R[1], \ldots, R[m]\}$,[4] the primary key can be logically expressed as the following set of formulas:
$$\forall \bar{x}\bar{y}\ (R(x_1, ..., x_m, x_{m+1}, \ldots, x_n) \wedge R(x_1, \ldots, x_m, y_{m+1}, \ldots, y_n) \rightarrow x_j = y_j),$$
$$\text{for } j = m + 1, \ldots, n.$$
$$\forall \bar{x}\bar{y}\ (R(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n) \wedge R(x_1, \ldots, x_m, y_{m+1}, \ldots, y_n) \wedge IsNull(x_j) \rightarrow IsNull(y_j)),$$
$$\text{for } j = m + 1, \ldots, n.$$
$$\forall \bar{x}\bar{y}\ (R(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n) \wedge IsNull(x_j) \rightarrow \textbf{false}), \qquad \text{for } j = 1, \ldots, m.$$

The third set of rules, are NNCs for all the attributes in the key. A unique constraint can be logically expressed by using only the first two set of rules. $\square$

---

[4]Without loss of generality we will assume the primary key to be the first m attributes of R

Now, by defining a primary key as in Definition 4.10, the problem presented in Example 4.15 is solved, and our semantics coincides with the one of SQL. Note that we are assuming that our database is a set of atoms and therefore, that there are no repeated atoms. If we defined instead a database instance as a bag of tuples, then our semantics would not coincide with SQL, as the following example shows.

Example 4.16 If we consider a database instance to be a bag of tuples, we could have the following database instance $D = \{P(a,b),\ P(a,b)\}$. If $P[1]$ is the primary key of P, the database would not be accepted as a consistent state in a SQL database. Our semantics is not able to distinguish between the two tuples and would, therefore consider it as consistent. $\qquad\square$

Theorem 4.1 Given a database instance D with no repeated tuples and set IC of primary keys, unique constraints, foreign keys, NOT NULL and check constraints, the database D satisfies IC with respect to the SQL standard (using simple-matching for foreign keys and no assertions) [International Organization for Standardization, 2003] iff $D \models_N IC$. $\qquad\square$

This theorem can be proved by appealing to the different constraints in the SQL standard [International Organization for Standardization, 2003] and to the corresponding notions of satisfaction in SQL for them [Türker and Gertz, 2001].

Our semantics of IC satisfaction for databases with *null* allows us to integrate our results in a compatible way with current commercial implementations; in the sense that the database repairs we will introduce later on would be accepted as consistent by current commercial implementations (for the classes of constraints that can be defined and maintained by them).

4.2 Query Answering in Databases with Null Values

With the purpose of answering first-order queries, we would like to extend our semantics of IC satisfaction with null values to query satisfaction, in an homogenous way.

We will assume, without loss of generality, that all the quantifiers in a first-order query are over different variables. For example, the query $\forall x P(x, y) \wedge \forall x Q(y, x)$, that has two quantifiers with variable x, can be transformed into the equivalent query $\forall x P(x, y) \wedge \forall z Q(y, z)$.

The queries may contain also the special predicate *IsNull*. This predicate will allow us to write SQL queries with `IS NULL` and `IS NOT NULL` expressions in first-order logic.

Example 4.17 Consider a table $P(A, B)$ and the SQL query:

```
SELECT P.A

FROM P

WHERE B IS NULL
```

The query can be written in first-order as $\exists y (P(x, y) \wedge \mathit{IsNull}(y))$. ☐

Definition 4.11 The set of *restricted relevant variables* of a first-order query ψ are:
$\mathcal{V}^R(\psi) = \{x \mid x$ is present at least twice in ψ, except for the variables in the *IsNull*
predicate $\}$ ☐

Example 4.18 For query $Q_1 : (P(x, y) \wedge \mathit{IsNull}(x) \wedge y > 5)$, $\mathcal{V}^R(Q_1) = \{y\}$ since y is used twice. Variable x is not repeated, since it appears only once in a predicate different from *IsNull*. For query $Q_2 : (P(x, y) \wedge Q(y, z) \wedge \mathit{IsNull}(y))$ the restricted relevant variables are $\mathcal{V}^R(Q_2) = \{y\}$. ☐

If the first-order query is a sentence (a boolean query) that expresses an integrity constraint, then Definition 4.11 of restricted relevant attributes is equivalent to Definition 4.8.

Definition 4.12 A *variable assignment function* $\mathbf{s}$ is a function from the set of variables to $\mathcal{U}$, the underlying database domain. We denote with $s[x|a]$ the assignment obtained from $\mathbf{s}$ by setting $s(x)$ to take the value a. A *term assignment function*, $\bar{s}$, associated to the variable assignment function $\mathbf{s}$, is defined as follows: (a) If term t is a variable x, then $\bar{s}(t) = s(x)$. (b) If t is a constant c or *null*, then $\bar{s}(t) = c$.

Given a formula ϕ, $\phi[\mathbf{s}]$ denotes the formula obtained from ϕ by replacing its free variables by its value according to $\mathbf{s}$. $\qquad\qquad\square$

Given a variable assignment function $\mathbf{s}$, we can check if D satisfies $\phi[\mathbf{s}]$ by assuming that a quantifier over a relevant variable is evaluated over $(\mathcal{U} \smallsetminus \{null\})$ and a non-relevant variable is evaluated over $\mathcal{U}$.[5] Formally:

Definition 4.13 Let ϕ be a first-order formula, $\mathbf{s}$ a variable assignment function and $\mathcal{B} = \{<, >, =, \mathbf{false}\}$. We define, by induction on ϕ, when D satisfies ϕ with assignment $\mathbf{s}$ with respect to the null-value semantics, denoted $D \models^q_N \phi[\mathbf{s}]$. Then, $D \models^q_N \phi[\mathbf{s}]$ when ϕ is of one of the following forms:

1. *IsNull*(t) and $\bar{\mathbf{s}}(t) = $ *null*.

2. $t_1 \diamond t_n$ for $\diamond \in \{<, >, =\}$, $\bar{\mathbf{s}}(t_1) \neq$ *null*, $\bar{\mathbf{s}}(t_2) \neq$ *null*, and $D \models t_1 \diamond t_n$.

3. $R(t_1, \ldots, t_n)$, with $R \in \mathcal{R}$ and $R(\bar{\mathbf{s}}(t_1), \ldots, \bar{\mathbf{s}}(t_n)) \in D$

4. $\neg\alpha$, and $D \not\models^q_N \alpha[\bar{\mathbf{s}}]$.

5. $(\alpha \vee \beta)$, and $D \models^q_N \alpha[\bar{\mathbf{s}}]$ or $D \models^q_N \beta[\bar{\mathbf{s}}]$

---

[5]By definition *null* $\in \mathcal{U}$.

6. $(\alpha \wedge \beta)$, and $D \models^q_N \alpha[\bar{s}]$ and $D \models^q_N \beta[\bar{s}]$

7. $(\forall y)(\alpha)$, and one of the following holds:

    (a) $y \in \mathcal{V}^R(\alpha)$, and for all a in $(\mathcal{U} \smallsetminus \{null\})$, $D \models^q_N \alpha[\bar{s}[y|a]]$.

    (b) $y \notin \mathcal{V}^R(\alpha)$, and for all a in $\mathcal{U}$, $D \models^q_N \alpha[\bar{s}[y|a]]$.

8. $(\exists y)(\alpha)$, and one of the following holds:

    (a) $y \in \mathcal{V}^R(\alpha)$, and there exists an a in $(\mathcal{U} \smallsetminus \{null\})$ with $D \models^q_N \alpha[\bar{s}[y|a]]$.

    (b) $y \notin \mathcal{V}^R(\alpha)$, and there exists an a in $\mathcal{U}$ with $D \models^q_N \alpha[\bar{s}[y|a]]$.

For all database instance D, $D \not\models^q_N \mathbf{false}$.   □

In the definition we assume $\mathcal{B} = \{<, >, =, \mathbf{false}\}$. The addition of new built-ins might modify the definition of relevant attribute.

Definition 4.14 A variable assignment $\mathbf{s}$ is *null-valid* with respect to ϕ if for every relevant variable x in ϕ, $\mathbf{s}(x) \neq null$.   □

Definition 4.15 (a) A tuple $(t_1, \ldots, t_n)$ with values in $\mathcal{U}$ is an answer from a database D under the *null query answering semantics* to a FOL query Q with free variables $(x_1, \ldots, x_n)$ iff there exists a null-valid assignment $\mathbf{s}$ for Q, such that $\mathbf{s}(x_i) = t_i$, for $i = 1, \ldots, n$, and $D \models^q_N Q[\mathbf{s}]$. (b) $Ans^N(Q, D)$ denotes the set of answers to Q obtained from database D under the semantics in (a). (c) If Q is a sentence (boolean query), the answer under the null query answering semantics is *yes* iff $D \models^q_N Q$ and *no* otherwise.   □

Example 4.19 Consider a database $D = \{P(f, 7), P(f, 5), P(d, 9), P(e, 2), P(null, 8), P(b, null)\}$ and the query $Q : \exists y(P(x, y) \wedge y > 5)$.

For this query $\mathcal{V}^R(Q) = \{y\}$ and the only free variable is x. Since x is not a relevant variable, a null-valid assignment can assign any value in the domain to x (including *null*). Let us check if $D \models_N^q Q[\mathbf{s_1}]$ for an assignment $\mathbf{s_1}$ with $\mathbf{s_1}(x) = f$.

The database $D \models_N^q \exists y(P(f,y) \wedge y > 5)$ if there exists $a \in (\mathcal{U} \smallsetminus \{null\})$, such that $D \models_N^q (P(f,a) \wedge a > 5)$. This is *true* for $a = 7$. Therefore, $D \models_N^q Q[\mathbf{s_1}]$, and (f) is an answer to the query. Analogously, (d) and $(null)$ are answers to the query.

Now, let us check if $D \models_N^q Q[\mathbf{s_2}]$ for an assignment $\mathbf{s_4}$ with $\mathbf{s_2}(x) = b$. Database $D \models_N^q \exists y(P(b,y) \wedge y > 5)$ if there exists $a \in (\mathcal{U} \smallsetminus \{null\})$, such that $D \models_N^q (P(b,a) \wedge a > 5)$. This value does not exist, therefore (b) is not an answer. $\qquad\square$

In the following example, we will check if the query answering semantics coincides with the semantics of IC satisfaction.

Example 4.20 Consider database D and a RIC $\psi : \forall x(P(x) \to \exists y\, R(x,y))$.

$D:$

P	X
	b
	null

R	X	Y
	b	null
	e	c

For this example, $\mathcal{V}^R(\psi) = \mathcal{V}(\psi) = \{x\}$, and $\mathcal{A}(\psi) = \{P[1], R[1]\}$. Therefore, in order to check if $D \models_N \psi$, we need to prove: $D^{\mathcal{A}(\psi)} \models \forall x\, (P^{\mathcal{A}(\psi)}(x) \to (x = null \vee R^{\mathcal{A}(\psi)}(x)))$. This is true, therefore the constraint is satisfied.

On the other hand, we can consider ψ a boolean first-order query. If the answer to the query is true, i.e., $D \models_N^q \psi$, then the constraint is satisfied. We use the inductive definition of satisfaction:

- Since x is relevant, $D \models_N^q (\forall x P(x) \to \exists y R(x,y))$ iff for all $a \in (\mathcal{U} \smallsetminus \{null\})$, it holds that $D \models_N^q (P(a) \to \exists y R(a,y))$.

- $D \models_N^q P(a) \to \exists y R(a,y)$ iff $D \not\models_N^q P(a)$ or $D \models_N^q \exists y R(a,y)$.

- $D \not\models_N^q P(a)$ for all $a \neq b$ (remember that $a \neq null$).

- For $a = b$, we have $D \models_N^q P(b)$, and therefore, we have to check if $D \models_N^q$ $\exists y R(b, y)$. Since y is not relevant, it is sufficient to check if there exists $c \in \mathcal{U}$, such that $D \models_N R(b, c)$. This is true since $R(b, null) \in D$.

Therefore, the answer to the query ψ is *Yes*. As expected, the semantics for satisfaction of constraints coincides with the query answering semantics. $\quad\square$

Proposition 4.1 Given a database instance D and a general integrity constraint ψ, $D \models_N \psi$ (see Definition 4.4) iff $D \models_N^q \psi$ (see Definition 4.13). $\quad\square$

Proof: First we will prove that if $D \models_N \psi$, then $D \models_N^q \psi$. The general IC ψ is of the form (4.4). $D \models_N \psi$ implies that $D^{\mathcal{A}(\psi)} \models \psi^N$, where ψ^N:

$$\forall \bar{x}'((\bigwedge_{i=1}^{m} P_i^{\mathcal{A}(\psi)}(\bar{x}_i') \wedge \bigwedge_{k=1}^{a} x_j = null) \;\rightarrow\; (\bigvee_{v_j \in \mathcal{V}^R(\psi)} v_j = null \;\vee\; \bigvee_{l=1}^{b} x_l = null \;\vee$$
$$\bigvee_{j=1}^{n} Q_j^{\mathcal{A}(\psi)}(\bar{y}_j) \;\vee\; \varphi)), \tag{4.6}$$

By contradiction, we will assume that $D \not\models_N^q \psi$. This implies that there exists an assignment $\bar{c} = (c_1, \ldots, c_n)$ for $\bar{x} = (x_1, \ldots, x_n)$, such that the following two statements hold:

1. If $x_i \in \mathcal{V}^R(\psi)$, then $c_i \in (\mathcal{U} \setminus \{null\})$. Otherwise, $c_i \in \mathcal{U}$.

2. $D \models_N^q ((\bigwedge_{i=1}^{m} P_i(\bar{c}_i) \wedge \bigwedge_{k=1}^{a} IsNull(c_k))$ and $D \not\models_N^q (\bigvee_{l=1}^{b} IsNull(c_l) \vee \exists \bar{z} (\bigvee_{j=1}^{n} Q_j(\bar{c}_j, \bar{z}_j)) \vee \varphi[\bar{s}[\bar{x}|\bar{c}]])$

Database $D \not\models_N^q \exists \bar{z}(\bigvee_{j=1}^{n} Q_j(\bar{c}_j, \bar{z}_j) \vee \varphi))$ implies that $D \not\models_N^q \bigvee_{l=1}^{b} IsNull(c_l)$, $D \not\models_N^q$ $\exists \bar{z} (\bigvee_{j=1}^{n} Q_j(\bar{c}_j, \bar{z}_j))$ and $D \not\models_N^q \varphi[\bar{s}[\bar{x}|\bar{c}]]$. Since no variable in $\bar{z}$ is relevant, $D \not\models_N^q \exists \bar{z}$ $(\bigvee_{j=1}^{n} Q_j(\bar{c}_j, \bar{z}_j))$ implies that, for all $\bar{d} \in \mathcal{U}$, $D \not\models_N^q \exists \bar{z} (\bigvee_{j=1}^{n} Q_j(\bar{c}_j, \bar{d}_j))$.

If we use assignment $\bar{c}$ in Equation (4.6), it follows that $v_j = null$ is false for every $v_j \in \mathcal{V}^R(\psi)$. This implies that $D^{\mathcal{A}(\psi)} \models (\bigvee_{l=1}^{b} c_l = null \vee \bigvee_{j=1}^{n} Q_j^{\mathcal{A}(\psi)}(\bar{c}_j) \vee \varphi[\bar{s}[\bar{x}|\bar{c}]])$, and therefore that $D^{\mathcal{A}(\psi)} \models \bigvee_{l=1}^{b} c_l = null$, $D^{\mathcal{A}(\psi)} \models \bigvee_{j=1}^{n} Q_j^{\mathcal{A}(\psi)}(\bar{c}_j)$ or $D^{\mathcal{A}(\psi)} \models \varphi[\bar{s}[\bar{x}|\bar{c}]]$. Then, $D \models \bigvee_{l=1}^{b} c_l = null$, $D \models \exists \bar{z} \bigvee_{j=1}^{n} Q_j(\bar{c}_j, \bar{z}_j))$ or $D \models \varphi[\bar{s}[\bar{x}|\bar{c}]]$. This contradicts the fact that $D \not\models_N^q \bigvee_{l=1}^{b} IsNull(c_l)$, $D \not\models_N^q \bigvee_{j=1}^{n} Q_j(\bar{c}_j, \bar{d}_j)$ and $D \not\models_N^q \varphi[\bar{s}[\bar{x}|\bar{c}]]$.

Now, let us prove that if $D \models_N^q \psi$, then $D \models_N \psi$. Let us assume by contradiction that $D \not\models_N \psi$. Database $D \not\models_N \psi$ implies that $D^{\mathcal{A}(\psi)} \not\models \psi^N$, where ψ^N:

$$\forall \bar{x}'((\bigwedge_{i=1}^{m} P_i^{\mathcal{A}(\psi)}(\bar{x}_i') \wedge \bigwedge_{k=1}^{a} x_k = null) \;\rightarrow\; (\bigvee_{v \in \mathcal{V}^R(\psi)} v = null \vee \bigvee_{l=1}^{b} x_l = null \vee$$
$$\bigvee_{j=1}^{n} Q_j^{\mathcal{A}(\psi)}(\bar{y}_j) \vee \varphi)), \tag{4.7}$$

Then there exists an assignment $s = s[\bar{x}' \,|\, \bar{c}]$ such that:[6]

$$D^{\mathcal{A}(\psi)} \models (\bigwedge_{i=1}^{m} P_i^{\mathcal{A}(\psi)}(\bar{c}_i') \wedge \bigwedge_{k=1}^{a} c_k = null) \tag{4.8}$$

$$D^{\mathcal{A}(\psi)} \not\models ((\bigvee_{v \in \mathcal{V}^R(\psi)} v = null)[\bar{s}] \vee \bigvee_{l=1}^{b} c_l = null \vee \bigvee_{j=1}^{n} Q_j^{\mathcal{A}(\psi)}(\bar{c}_j) \vee \varphi[\bar{s}]) \tag{4.9}$$

It follows from equation (4.8) that there exists an extension of the variable assignment such that $D \models (\bigwedge_{i=1}^{m} P_i(\bar{c}_i) \wedge \bigwedge_{k=1}^{a} c_k = null)$. It follows from equation (4.9) that, for every $v \in \mathcal{V}^R(\psi)$, $s(v) \in (\mathcal{U} \smallsetminus \{null\})$. Then, it follows from $D \models (\bigwedge_{i=1}^{m} P_i(\bar{c}_i) \wedge \bigwedge_{k=1}^{a} c_k = null)$ that $D \models_N^q (\bigwedge_{i=1}^{m} P_i(\bar{c}_i) \wedge \bigwedge_{k=1}^{a} c_k = null)$. Then, since $D \models_N^q \psi$, there exists $s' = s[\bar{z} \,|\, \bar{d}]$ such that $D \models_N^q (\bigvee_{l=1}^{b} c_l = null \vee \bigvee_{j=1}^{n} Q_j(\bar{c}_j, \bar{d}_j) \vee \varphi[\bar{s}'])$. As a consequence, $D \models_N^q (\bigvee_{l=1}^{b} c_l = null)$, $D \models_N^q (\bigvee_{j=1}^{n} Q_j(\bar{c}_j, \bar{d}_j))$ or $D \models_N^q (\varphi[\bar{s}'])$.

---

[6]Note that $y_j \subseteq \bar{x}'$, and therefore that an assignment for variables in $\bar{x}'$ also assigns values to variables in each y_j.

This is inconsistent with equation (4.9). □

4.2.1   Relationship with SQL

Query answering in SQL is not an extension of the semantics of IC satisfaction in SQL standard, and therefore, any uniform extension of IC satisfaction semantics (like the null query answering semantics presented in the previous section) will not coincide with SQL for some types of queries.

Example 4.21 (example 4.20 continued) Consider the query $Q : (P(x) \wedge \neg \exists y R(x, y))$. Since S satisfies ψ, i.e., $D \models_N \forall x (P(x) \rightarrow \exists y \ R(x, y))$, we would expect to have *no* answers to query Q. For the null query answering semantics it holds $Ans^N(Q, D) = \emptyset$, which is consistent with the fact that ψ is satisfied by D.

Query Q can be rewritten as a SQL query Q^{SQL}:

```
SELECT  X
FROM    P
WHERE   NOT EXISTS   (SELECT *
                      FROM   R
                      WHERE  R.X=P.X)
```

The answer to this query in SQL is not empty, but $\{(null)\}$. Even though the constraint is satisfied in SQL, the answer to query Q in SQL is not empty. □

Example 4.22 (example 4.19 continued) Query Q can be rewritten as the following SQL query Q^{SQL}:

```
SELECT  X
FROM    P
WHERE   Y>5
```

If this SQL query is posed on database D, it will return the set of tuples $\{(f), (d), (null)\}$. In this case, the answers obtained from a SQL query coincide with our query answering semantics. □

Example 4.23 Consider database D and the SQL query below.

D :

P	A	B
	b	1
	c	$null$

T	C	D
	b	3
	e	1

```
SELECT  A
FROM    P
WHERE   NOT EXISTS  (SELECT *
                     FROM    T
                     WHERE   T.D=P.B)
```

The only answer to the SQL query is (c). The SQL query can be written in FOL as $Q : \exists y(P(x, y) \wedge \neg(\exists z T(z, y)))$. The set of restricted relevant variables is $\mathcal{V}^R(Q) = \{y\}$ and the only free variable is x. Then, a null-valid assignment s will be such that $s(x) \in \mathcal{U}$. Let us first take $s(x) = c$. Now, $D \models^q_N \exists y(P(c, y) \wedge \neg(\exists z T(z, y)))$ if there exists a constant $k \in (\mathcal{U} \smallsetminus \{null\})$ such that $D \models^q_N (P(c, k) \wedge \neg(\exists z T(z, k)))$. There is no such k; therefore (c) is not an answer to the query. In fact, the set of answers to query Q is empty. In this case, the answers given by SQL and the null query answering semantics do not coincide. □

Definition 4.16 Given a SQL query Q and a database D, $Ans^{SQL}(Q, D)$ is the set of answers obtained by executing Q on D using the *SQL query answer semantics* [International Organization for Standardization, 2003]. For a tuple $\bar{t}$, we also denote $\bar{t} \in Ans^{SQL}(Q, D)$ with $D \models^{SQL}_N Q[\bar{t}]$. □

A first-order *conjunctive query* (CQ) is of the form : $Q : \exists \bar{y}(\bigwedge_{i=1}^{n} P_i(\bar{x}_i, \bar{y}_i) \wedge \varphi)$, where the P_i are database predicates, $\bar{y} = \cup_{i=1}^{n} \bar{y}_i$, and φ is a conjunction of comparison atoms. Let $\{x_1, \ldots, x_m\} = \cup_{i=1}^{n} \bar{x}_i$. A CQ can be rewritten as a SQL query Q^{SQL} as follows :

    SELECT $x'_1, \ldots, x'_m$
    FROM $P_1, \ldots, P_n$
    WHERE $P_i.v = P_j.v$, AND ... AND $P_l.u = P_k.u$, AND φ^{SQL}

where x'_i is an attribute associated with variable x_i in Q, the conditions in the WHERE represent the joins, and φ^{SQL} replaces each variable x in φ by its x' version.

Example 4.24 Consider relations $T(A, B)$ and $S(C, D, E)$, and the conjunctive query $Q : \exists yz(T(x, y) \wedge S(y, z, u) \wedge u > 5)$. This query can be written in SQL as Q^{SQL}:

    SELECT A, E
    FROM   T, S
    WHERE   T.B = S.C AND E > 5   □

These queries are very common in database praxis. It can be shown that the answers to a query of this kind obtained from both the null query answering semantics and the SQL query answering semantics coincide.

Proposition 4.2 For a CQ Q, $Ans^N(Q, D) = Ans^{SQL}(Q^{SQL}, D)$.   □

Besides the class CQ, other relevant queries are the first-order conjunctive queries with negation. Negation in SQL queries can be expressed using the NOT EXISTS statement.

Example 4.25 For relations $P(A, B)$, $T(C, D)$ and $S(E)$, the query $Q_1 : \exists y P(x, y) \wedge \neg \exists z T(x, z) \wedge \neg S(y)$ can be written as Q_1^{SQL} in SQL as follows:

```
SELECT  A

FROM    P

WHERE   NOT EXISTS   (SELECT *  FROM T  WHERE  T.C=P.A)

        AND

        NOT EXISTS   (SELECT *  FROM S  WHERE  S.E=P.B)
```

Query $Q_2 : P(x,y) \wedge \neg \exists z(T(x,z) \wedge \neg S(z))$ can be rewritten as Q_2^{SQL} into an SQL query:

```
SELECT  A

FROM    P

WHERE   NOT EXISTS   (SELECT *

                      FROM T

                      WHERE  T.C=P.A

                             AND

                             NOT EXISTS (SELECT *

                                         FROM S

                                         WHERE S.E=T.D))          □
```

We will now define a set of conjunctive queries Q with negation for which we can define a rewriting Q', such that the answers obtained for Q' under the null query answering semantics coincide with those obtained from Q for the SQL query answering semantics. This set of queries consists of queries that can be rewritten in SQL without nested occurrences of NOT EXISTS. We leave the analysis of queries with nested NOT EXISTS for future research.

Definition 4.17 An *extended conjunctive queries* (ECQ) is of the form

$$\exists \bar{v}(Q_0 \wedge \bigwedge_{j=1}^{n} \neg Q_j)$$

where Q_0, Q_1, ... and Q_n are conjunctive queries; and each variable $v \in \bar{v}$ is a free variable in Q_0, and every free variable of $Q_1, \ldots, Q_n$ is also in Q_0. □

Extended conjunctive queries are domain independent and can be rewritten as SQL queries using the NOT EXISTS SQL statement. If $n = 0$, the query is a conjunctive query.

Example 4.26 (example 4.25 continued) Q_1 is an extended conjunctive query, but Q_2 is not since it has nested NOT EXISTS expressions. □

Example 4.27 The queries $Q_1 : (P(x, y) \wedge R(y, z))$ and $Q_2 : \exists u(T(y, u) \wedge u > 3)$ are both CQ and ECQ. Query $Q_3 : \exists y(Q_1 \wedge \neg Q_2) = \exists y \ (P(x, y) \wedge R(y, z) \wedge \neg(\exists u T(y, u) \wedge u > 3))$ is by construction it is a ECQ, but not a CQ. In the case of query $Q_4 : \exists yu(P(x, y) \wedge \neg T(x, u))$, variable u is in a negated subquery, and is not in the positive part, therefore Q_4 is not a ECQ. It is not a CQ either since it has negation. □

The following example shows that for extended conjunctive queries, it may happen that $Ans^N(Q, D) \neq Ans^{SQL}(Q, D)$.

Example 4.28 Consider a database D and the query $Q_1(x, z) : \exists y \ (P(x, y) \wedge R(y, z) \wedge \neg(\exists u R(z, u)))$.

D :

P	A	B
	a	b
	c	c
	a	g
	d	$null$

R	C	D
	b	c
	c	$null$
	g	b

The SQL version Q_1^{SQL} of this ECQ query is:

```
SELECT A D

FROM P, R AS R1

WHERE  P.B=R1.C AND NOT EXISTS  (SELECT *

                                 FROM R AS R2

                                 WHERE R1.D=R2.C)
```

In this case, $Ans^N(Q_1, D) = \{(a, g)\} \neq Ans^{SQL}(Q_1^{SQL}, D) = \{(c, null), (a, g)\}$.   □

Example 4.29 (example 4.28 continued) Consider $Q_2(x) : \exists y \ (P(x, y) \wedge \neg \exists u(R(y, u) \wedge R(c, y)))$. The SQL version Q_2^{SQL} of this ECQ query is:

```
SELECT A

FROM P

WHERE  NOT EXISTS  (SELECT *

                    FROM R AS R1, R AS R2

                    WHERE R1.A=R2.D)
```

In this case, $Ans^N(Q_2, D) = \emptyset \neq Ans^{SQL}(Q_2^{SQL}, D) = \{(d)\}$.   □

There are some other cases in which $Ans^N(Q, D)$ and $Ans^{SQL}(Q^{SQL}, D)$ coincide.

Example 4.30 (example 4.28 continued) Consider $Q_3(x, z) : \exists y \ (P(x, y) \wedge R(y, z) \wedge \neg(\exists u R(y, u)))$. The SQL version Q_3^{SQL} of this ECQ query is:

```
SELECT A

FROM P, R AS R1

WHERE  P.B=R1.C AND NOT EXISTS  (SELECT *

                                 FROM R AS R2

                                 WHERE R1.C=R2.C)
```

In this case, $Ans^N(Q_3, D) = Ans^{SQL}(Q_3^{SQL}, D) = \{(a, b)\}$.   □

Definition 4.18 Given an ECQ Q of the form $Q = \exists \bar{v}(Q_0 \wedge \bigwedge_{j=1}^{n} \neg Q_j)$, the *positive relevant attributes of Q* are $\mathcal{A}^+(Q) = \mathcal{A}(Q_0)$. Let $\mathcal{A}^-(Q) = \mathcal{A}(Q) \setminus \mathcal{A}^+(Q)$. $\square$

It can be checked in the three previous examples that, when $\mathcal{A}^-(Q) \neq \emptyset$, the difference between $Ans^N(Q, D)$ and $Ans^{SQL}(Q^{SQL}, D)$ can be attributed to not assigning *null* to one of the variables in $\mathcal{A}^-(Q)$.

Definition 4.19 Given an ECQ Q of the form $Q = \exists \bar{v}(Q_0 \wedge \bigwedge_{j=1}^{n} \neg Q_j)$, the rewritten query

$$Q^{\mathfrak{N}} = Q \vee \bigvee_{e \in \mathcal{E} \wedge e \neq \emptyset} (Q_1[\bar{s}_e] \wedge \bigwedge_{Q_j \in \mathcal{Q}_e} \neg Q_j),$$

where s_e is a substitution that assigns each variable $v \in e$ to *null*, $\mathcal{E}$ is the powerset of $\mathcal{A}^-(Q)$ and $\mathcal{Q}_e = \{Q_j \mid j \in \{1, \dots, n\}$ and no free variable in Q_j is in set $e\}$. $\square$

Note that if Q is a CQ, then $\mathcal{A}^-(Q) = \emptyset$ and $Q^{\mathfrak{N}} = Q$.

Example 4.31 (examples 4.28, 4.29, 4.30 continued) Query Q_1 has $\mathcal{A}^-(Q_1) = \{z\}$ and $\mathcal{E} = \{\emptyset, \{z\}\}$. Therefore, $Q_1^{\mathfrak{N}}(x, z) : (\exists y\ P(x, y) \wedge R(y, z) \wedge \neg\ \exists u R(z, u)) \vee (\exists y\ (P(x, y) \wedge R(y, null)))$. Query Q_2 has $\mathcal{A}^-(Q_2) = \{y\}$ and $\mathcal{E} = \{\emptyset, \{y\}\}$; and therefore $Q_2^{\mathfrak{N}}(x) : (\exists y\ P(x, y) \wedge \neg\ \exists u(R(y, u) \wedge R(c, y))) \vee \exists y\ P(x, null)$. Query Q_3 has $\mathcal{A}^-(Q_3) = \emptyset$ and $\mathcal{E} = \{\emptyset\}$, and therefore $Q_3^{\mathfrak{N}}(x, z) : \exists y\ (P(x, y) \wedge R(y, z) \wedge \neg\ \exists u\ R(y, u))$.

Consider now the ECQ $Q_4(x, y) : T(x, y, z) \wedge y > 3 \wedge \neg\ \exists v\ R(x, v) \wedge \neg\ S(y, z)$. For this query, $\mathcal{A}^-(Q_4) = \{x, z\}$ and $\mathcal{E} = \{\emptyset, \{x\}, \{z\}, \{x, z\}\}$, and $Q_4^{\mathfrak{N}}(x, y) : (T(x, y, z) \wedge y > 3 \wedge \neg\ \exists v\ R(x, v) \wedge \neg\ S(y, z)) \vee (T(null, y, z) \wedge y > 3 \wedge \neg\ S(y, z)) \vee (T(x, y, null) \wedge y > 3 \wedge \neg\ \exists v\ R(x, v)) \vee (T(null, y, null) \wedge y > 3)$. $\square$

This rewriting of query Q allows us to get the answers that are missing from $Ans^N(Q, D)$, to make it equal to $Ans^{SQL}(Q^{SQL}, D)$.

Proposition 4.3 Given an ECQ Q and a database D:

$$Ans^N(Q^{\mathfrak{N}}, D) = Ans^{SQL}(Q^{SQL}, D).$$ □

Example 4.32 (example 4.28 and 4.31 continued) Considering the null query answering semantics, the answers obtained from $Q_1^{\mathfrak{N}}(x, z)$: $\exists y \ (P(x, y) \wedge R(y, z) \wedge \neg(\exists u R(z, u))) \vee (P(x, y) \wedge R(y, null))$ coincide with the answers of Q_1 under the SQL query answering semantics. That is, $Ans^N(Q_1^{\mathfrak{N}}, D) = Ans^{SQL}(Q_1^{SQL}, D) = \{(c, null), (a, g)\}$. □

The results for ECQ cannot be applied directly for SQL queries with nested `NOT EXISTS` statements.

Example 4.33 Consider a database D and a query $Q(x, y) : P(x, y) \wedge \neg\exists z(R(x, z) \wedge \neg\exists w S(z, w))$.

D :

P	A	B
	a	b

R	C	D
	a	$null$

S	E	F
	c	d

Query Q can be written is SQL as Q^{SQL}:

```
SELECT A B

FROM P

WHERE  NOT EXISTS  (SELECT *

                    FROM R

                    WHERE P.A=R.C AND

                          NOT EXISTS   (SELECT *

                                        FROM S

                                        WHERE R.D=S.E))
```

In this case, $Ans^N(Q, D) = \{(a, b)\}$ and $Ans^{SQL}(Q^{SQL}, D) = \emptyset$. If we try to use the

techniques for ECQ we get $Ans^N(Q^{\mathfrak{N}}, D) = \{(a, b)\}$, which does not coincide with $Ans^{SQL}(Q^{SQL}, D)$. □

It is left for future research find a rewriting for SQL queries with nested negation that can be used with the null query answering semantics.

These results just obtained are interesting in themselves, but we will also make crucial use of them in Section 5.2.1, when retrieving consistent answers to queries using the SQL query semantics. The consistent answers of Q will be obtained by first rewriting Q into $Q^{\mathfrak{N}}$, then producing a query program $\Pi(Q^{\mathfrak{N}})$; and finally computing the stable models of $\Pi(Q^{\mathfrak{N}})$ combined with a logic programming specification of repairs.

4.3 Conclusions

Motivated by the problem of consistent query answering in databases, as we find them in commercial DBMS implementations, that only partially conform to the SQL standard, we revisit the problems of integrity constraint satisfaction and query answering. More specifically, we have proposed a precise and uniform logical reconstruction of IC satisfaction for databases with only one kind of null value. This semantics is compatible with the way null values are treated according to the SQL standard. The integrity constraint satisfaction semantics presented in this chapter was published in [Bravo and Bertossi, 2006].

We also provide a semantics for query answering, called null query answering semantics, that extends the one for IC satisfaction, but that does not always coincide with the query answering semantics of SQL. However, the null and SQL query answering semantics coincide for conjunctive queries. For extended conjunctive queries they might not coincide, but if an ECQ Q is rewritten as Q^N, the answers to Q under the SQL query semantics coincide with the answers to Q^N under the null query

semantics. The advantage of being able to use the null query answering semantics is that it can be easily reduced to first-order query answering semantics of predicate logic.

The satisfaction of an integrity constraint can be checked using *violation view* [Gupta and Mumick, 1995], that contains the set of tuples that violate the constraint. If the violations view is empty, the IC is satisfied. Examples 4.20 and 4.21 show a case in which even though the constraint is satisfied, the traditional violation view is not empty. Therefore, in databases with *null* it is not possible to use the traditional violation views to check the satisfaction of constraints. It should be possible to provide a general methodology to (re)define violation views for databases with *null*.

For example, for the integrity constraint in Example 4.20, the following query would define the violation view for databases with null values, and can be used to correctly check the satisfaction of the IC under the SQL semantics:

```
SELECT   X
FROM     P
WHERE    NOT EXISTS    (SELECT *
                         FROM   R
                         WHERE  R.X=P.X)
         AND IS NOT NULL X
```

With respect to related work, there are plenty of different semantics for IC satisfaction and query answering in databases with null values. For a survey see [van der Meyden, 1998]. Integrity constraint satisfaction in the SQL standard is described in [International Organization for Standardization, 2003] (cf. [Türker and Gertz, 2001; Ullman and Widom, 2002]). There, for each type of constraint, a separate definition of satisfaction is given. So a uniform semantics is missing.

One of the most widely accepted semantics for incomplete databases is the *possible*

worlds semantics [Kripke, 1971; Imielinski and Lipski, 1984; Grahne, 1991]. This semantics interprets *null* as an unknown, but existing value. Since null values in commercial databases can also be interpreted as non-existent, this semantics does not coincide with the SQL semantics for satisfaction of constraints.

Semantics for IC satisfaction are introduced in [Grant, 1980; Levene and Loizou, 1997b; Atzeni and Morfuni, 1984; Atzeni and Morfuni, 1986; Lien, 1979]. They all consider a unique null value and consider some restricted types of constraints, such as functional dependencies and/or inclusion dependencies. None of those semantics gives an account of the behavior of commercial DBMSs. Finally, in [Buneman *et al.*, 1991; Libkin, 1991; Libkin, 1995; Reiter, 1984; Reiter, 1986] incomplete databases are studied for general type of constraints, but their semantics does not coincide with the one in commercial DBMSs either.

Chapter 5

CQA in Relational Databases

In Section 2.2, we review techniques to compute consistent query answers. The techniques are query rewriting [Arenas *et al.*, 1999; Celle and Bertossi, 2000], and specification of repairs using logic programs [Barceló and Bertossi, 2002; Barceló; and Bertossi, 2003]. The latter deals with constraints and queries that are more general than the ones dealt by the former. However, the repair specification is still not general enough to deal with the most common ICs such as foreign key constraints.

The repair programs in [Barceló and Bertossi, 2002; Barceló; and Bertossi, 2003] use annotation constants with the intended, informal semantics shown in Table 5.1 . The annotations are used in an extra attribute introduced in each database predicate; so for a predicate $P \in \mathcal{R}$, the new version of it, $P_{\text{-}}$, contains an extra attribute.

Definition 5.1 [Barceló and Bertossi, 2002] Given a database D and a set of UICs IC of the form (2.2), the *repair program* $\Pi(D, IC)$ is a disjunctive logic program formed by the following set of rules:

Annotation	Atom	The tuple $P(\bar{a})$ is...
$\mathbf{t_d}$	$P_{\text{-}}(\bar{a}, \mathbf{t_d})$	true in the database
$\mathbf{t_a}$	$P_{\text{-}}(\bar{a}, \mathbf{t_a})$	advised to be made true
$\mathbf{f_a}$	$P_{\text{-}}(\bar{a}, \mathbf{f_a})$	advised to be made false
$\mathbf{t^\star}$	$P_{\text{-}}(\bar{a}, \mathbf{t^\star})$	true or becomes true
$\mathbf{f^\star}$	$P(\bar{a}, \mathbf{f^\star})$	false or becomes false
$\mathbf{t^{\star\star}}$	$P_{\text{-}}(\bar{a}, \mathbf{t^{\star\star}})$	it is true in the repair
$\mathbf{f^{\star\star}}$	$P(\bar{a}, \mathbf{f^{\star\star}})$	it is false in the repair

Table 5.1: Annotation constants and their meaning

1. $dom(x)$ for every constant x in database D.

2. The fact $P_{\_}(\bar{a}, \mathbf{t_d})$ for every atom $P(\bar{a}) \in D$.

3. For every predicate $p \in \mathcal{R}$, $\Pi(D, IC)$ contains the rules:

$P_{\_}(\bar{x}, \mathbf{t}^\star) \leftarrow P_{\_}(\bar{x}, \mathbf{t_d}).$ $\qquad$ $P_{\_}(\bar{x}, \mathbf{t}^\star) \leftarrow P_{\_}(\bar{x}, \mathbf{t_a}).$

$P_{\_}(\bar{x}, \mathbf{f}^\star) \leftarrow P(\bar{x}, \mathbf{f_a}).$ $\qquad$ $P_{\_}(\bar{x}, \mathbf{f}^\star) \leftarrow dom(\bar{x}),\ not\ P_{\_}(\bar{x}, \mathbf{t_d}).$

4. For every constraint of the form (2.2), $\Pi(D, IC)$ contains the clause:

$$\bigvee_{i=1}^{n} P_{i\_}(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} Q_{j\_}(\bar{y}_j, \mathbf{t_a}) \ \longleftarrow\ \bigwedge_{i=1}^{n} P_{i\_}(\bar{x}_i, \mathbf{t}^\star) \wedge \bigwedge_{j=1}^{m} Q_{j\_}(\bar{y}_j, \mathbf{f}^\star) \wedge \bar{\varphi},$$

where $\bar{\varphi}$ represents the negation of φ.

5. For every predicate $p \in \mathcal{R}$, $\Pi(D, IC)$ contains the rules:

$P_{\_}(\bar{x}, \mathbf{f}^{\star\star}) \ \leftarrow\ P_{\_}(\bar{x}, \mathbf{f_a}).$ $\qquad$ $P_{\_}(\bar{x}, \mathbf{f}^{\star\star}) \ \leftarrow\ dom(\bar{x}),\ not\ P_{\_}(\bar{x}, \mathbf{t_d}),\ not\ P_{\_}(\bar{x}, \mathbf{t_a}).$

$P_{\_}(\bar{x}, \mathbf{t}^{\star\star}) \ \leftarrow\ P_{\_}(\bar{x}, \mathbf{t_a}).$ $\qquad$ $P_{\_}(\bar{x}, \mathbf{t}^{\star\star}) \ \leftarrow\ P_{\_}(\bar{x}, \mathbf{t_d}),\ not\ P_{\_}(\bar{x}, \mathbf{f_a}).$

6. For every predicate $p \in \mathcal{R}$, $\Pi(D, IC)$ contains the rules:

$\leftarrow\ P_{\_}(\bar{x}, \mathbf{t_a})\ ,\ P_{\_}(\bar{x}, \mathbf{f_a}).$ $\qquad\qquad\qquad\qquad\qquad$ $\square$

The rules in 1. correspond to database tuples. The rules in 2. correspond to the facts in the database. The rules in 3. collect with $\mathbf{t}^\star$ (respectively $\mathbf{f}^\star$ elements that are true (false) or have been made true (false). Rules in 4. are the most important and express how the inconsistencies should be repaired. The body of the rule will be true if the constraint is not satisfied and the head indicates how to repair the inconsistency. Rules in 5. collect with $\mathbf{t}^{\star\star}$ all the tuples that will be true in the repairs. Rule 6. enforces that a tuple cannot be made true and false at the same time.

Example 5.1 Consider a database that stores a table *Course* with the list of courses and a table *Reg* with the IDs of students registered in each course. Consider the following database instance D:

Course	Code		Reg	ID	Code
	C21			21	C15
	C15			34	C18

The primary keys are *ID* and *Code*[1] respectively, and there is a foreign key constraint from attribute *Code* in table *Reg* to the primary key of table *Course*. Database D is inconsistent with respect to this set of constraints since there is a *Code* in table *Reg* that is not in table *Course*.

The integrity constraints can be written in form (2.1) as follows:

$\forall xyz\, Reg(x,y) \land Reg(x,z) \to y = z.$

$\forall xy\, Reg(x,y) \to Course(y).$

Both of these constraints are universal integrity constraints of the form (2.2) and therefore we can use the repair program of Definition 5.1:

1. $Course\_(C21, \mathbf{t_d}).$ $Course\_(C15, \mathbf{t_d}).$

   $Reg\_(21, C15, \mathbf{t_d}).$ $Reg\_(34, C18, \mathbf{t_d}).$

2. $Course\_(x, \mathbf{t^\star}) \leftarrow Course\_(x, \mathbf{t_d}).$ $Course\_(x, \mathbf{t^\star}) \leftarrow Course\_(x, \mathbf{t_a}).$

   $Course\_(x, \mathbf{f^\star}) \leftarrow Course(x, \mathbf{f_a}).$ $Course\_(x, \mathbf{f^\star}) \leftarrow \ not\ Course\_(x, \mathbf{t_d}).,$

   $Reg\_(x, y, \mathbf{t^\star}) \leftarrow Reg\_(x, y, \mathbf{t_d}).$ $Reg\_(x, y, \mathbf{t^\star}) \leftarrow Reg\_(x, y, \mathbf{t_a}).$

   $Reg\_(x, y, \mathbf{f^\star}) \leftarrow Reg(x, y, \mathbf{f_a}).$ $Reg\_(x, y, \mathbf{f^\star}) \leftarrow \ not\ Reg\_(x, y, \mathbf{t_d}).,$

3. $Reg\_(x, y, \mathbf{f_a}) \lor Reg\_(x, z, \mathbf{f_a}) \leftarrow Reg\_(x, y, \mathbf{t^\star}), Reg\_(x, z, \mathbf{t^\star}), y \neq z.$

   $Reg\_(x, y, \mathbf{f_a}) \lor Course\_(y, \mathbf{t_a}) \leftarrow Reg\_(x, y, \mathbf{t^\star}), Course\_(y, \mathbf{f^\star}).$

4. $Course\_(x, \mathbf{f^{\star\star}}) \leftarrow Course\_(x, \mathbf{f_a}).$

   $Course\_(x, \mathbf{f^{\star\star}}) \leftarrow \ not\ Course\_(x, \mathbf{t_d}),\ not\ Course\_(x, \mathbf{t_a}).$

[1]Note that since *Code* is the only attribute of table *Course*, the primary key cannot be violated.

$$Course\_(x, \mathbf{t}^{\star\star}) \leftarrow Course\_(x, \mathbf{t_a}).$$

$$Course\_(x, \mathbf{t}^{\star\star}) \leftarrow Course\_(x, \mathbf{t_d}), \ not \ Course\_(x, \mathbf{f_a}).$$

$$Reg\_(x, y, \mathbf{f}^{\star\star}) \leftarrow Reg\_(x, y, \mathbf{f_a}).$$

$$Reg\_(x, y, \mathbf{f}^{\star\star}) \leftarrow not \ Reg\_(x, y, \mathbf{t_d}), \ not \ Reg\_(x, y, \mathbf{t_a}).$$

$$Reg\_(x, y, \mathbf{t}^{\star\star}) \leftarrow Reg\_(x, y, \mathbf{t_a}).$$

$$Reg\_(x, y, \mathbf{t}^{\star\star}) \leftarrow Reg\_(x, y, \mathbf{t_d}), \ not \ Reg\_(x, y, \mathbf{f_a}).$$

5. $\leftarrow Course\_(x, \mathbf{t_a}), Course\_(x, \mathbf{f_a}).$

$\leftarrow Reg\_(x, y, \mathbf{t_a}), Reg\_(x, y, \mathbf{f_a}).$

The most important rules here are the rules in 3. The first one specifies the requirements on a repair if the primary key of table *Reg* is violated, and the second one if the foreign key is violated. If we compute the stable models of this program and select only the tuples annotated with $\mathbf{t}^{\star\star}$, we get the following sets:

$$\mathcal{M}_1 = \{Course\_(C21, \mathbf{t}^{\star\star}), \ Course\_(C15, \mathbf{t}^{\star\star}), \ Course\_(C18, \mathbf{t}^{\star\star}), \ Reg\_(21, C15, \mathbf{t}^{\star\star})\}$$

$$\mathcal{M}_2 = \{Course\_(C21, \mathbf{t}^{\star\star}), \ Course\_(C15, \mathbf{t}^{\star\star}), \ Reg\_(21, C15, \mathbf{t}^{\star\star}), \ Reg\_(34, C18, \mathbf{t}^{\star\star})\}$$

Which correspond to the following two repairs:

D_1:

Course	Code
	C21
	C15

Reg	ID	Code
	21	C15

D_2 :

Course	Code
	C21
	C15
	C18

Reg	ID	Code
	21	C15
	34	C18

□

The repair program of Definition 5.1 can be used to obtain the repairs only for UICs.

There are important types of constraints that are not considered, e.g. not all foreign key constraints can be expressed as UICs. This is why we want to extend the repair programs to deal with RICs, i.e., constraints of the form (2.3).

Example 5.2 (example 2.5 continued). In this example, there is an inclusion dependency that references only attribute *ID* from table *Student* and therefore *Name* is existentially quantified in the integrity constraints. Therefore, this constraint is not a UIC, but a RIC. We can also observe that the number of repairs in this case, using the repair semantics of [Arenas *et al.*, 1999], is infinite. □

In [Calì *et al.*, 2003a], it was proven that the problem of CQA for a cyclic set of constraints with existential quantifiers is undecidable for the repair semantics in [Arenas *et al.*, 1999]. This implies that we cannot implement the repair semantics for RICs. In [Barceló; and Bertossi, 2003], a variation of the repair semantics of [Arenas *et al.*, 1999] is proposed but not formalized. There, for a RIC of the form $P(\bar{x}) \to \exists y(Q(\bar{x}', y))$, where $\bar{x}' \subseteq \bar{x}$, it is suggested to repair by inserting *null*, say $Q(\bar{a}, null)$, or by deletion. Taking this variation into consideration we can redefine repairs.

Example 5.3 (example 2.5 and 5.2 continued) If, instead of repairing a database with respect to a RIC by inserting all the possible values in the domain, we could repair by adding *null* in the position of the existentially quantified values, then, there would only be two repairs for this example:

D' :

Reg	ID	Code
	21	C15

Student	ID	Name
	21	Ann
	45	Paul

D'' :

Reg	ID	Code
	21	C15
	34	C18

Student	ID	Name
	21	Ann
	45	Paul
	34	null

□

In the next section, we formalize this new repair semantics.

5.1 Repairing using *null*

Now, not only databases may contain *null*, but they can be used to repair inconsistencies. In order to define formally new semantics for repairs, we need a new definition of distance that captures the fact that we prefer to repair with *null* than with an arbitrary value of the domain.

Definition 5.2 Let D, D', D'' be database instances over the same schema and domain $\mathcal{U}$. It holds that $D' \leq_D D''$ iff for every database atom $P(\bar{a}) \in \Delta(D, D')$, there exists an atom $P(\bar{a}')$, such that: (a) $P(\bar{a}') \in \Delta(D, D'')$; (b) $P(\bar{a}) \sqsubseteq P(\bar{a}')$;[2] and (c) if $P(\bar{a}) \sqsubset P(\bar{a}')$, then $P(\bar{a}') \notin \Delta(D, D')$. □

Now, a repair is a new database, possibly with *null*, that satisfies the ICs as defined in Section 4.1 and that solves inconsistencies with respect to RICs by inserting *null*.

Definition 5.3 Given a database instance D and a set IC of ICs of the form (2.1) and NNCs, a repair of D with respect to IC is a database instance D' over the same schema, such that $D' \models_N IC$ and D' is $\leq_D$-*minimal* in the class of database instances that satisfy IC with respect to $\models_N$, and share the schema with D, i.e., there is no database D'' in this class with $D'' <_D D'$, where $D'' <_D D'$ means $D'' \leq_D D'$ but not $D' \leq_D D''$. We denote by $Rep(D, IC)$ the set of repairs of D with respect to IC. □

---

[2]For the definition of $\sqsubseteq$ see Definition 4.1.

For a database without *null* and a set of UICs, this definition of repair coincides with the one in [Arenas *et al.*, 1999].

Example 5.4 The database instance $D = \{Q(a,b), P(a,c)\}$ is inconsistent with respect to the ICs $\psi_1 : (P(x,y) \rightarrow \exists z Q(x,z))$ and $\psi_2 : (Q(x,y) \rightarrow y \neq b)$, because $D \not\models_N \psi_2$. The database has two repairs with respect to $\{\psi_1, \psi_2\}$, namely $D_1 = \{\}$, with $\Delta(D, D_1) = \{Q(a,b), P(a,c)\}$, and $D_2 = \{P(a,b), Q(a, null))\}$, with $\Delta(D, D_2) = \{Q(a,b), Q(a, null)\}$. Notice that $D_2 \not\leq_D D_1$, because for $Q(a, null) \in \Delta(D, D_2)$, even though $Q(a,b) \in \Delta(D, D_1)$ is such that $Q(a, null) \sqsubset Q(a,b)$, $Q(a,b)$ is also in $\Delta(D, D_2)$ violating condition (c) of Definition 5.2. Similarly, $D_1 \not\leq_D D_2$, because $P(a,c) \in \Delta(D, D_1)$ and $P(a,c) \notin \Delta(D, D_1)$. □

Example 5.5 If the database instance is $\{P(a, null), P(b,c), R(a,b)\}$ and the set *IC* consists only of $(P(x,y) \rightarrow \exists z \ R(x, z))$, then there are two repairs: $D_1 = \{P(a, null), P(b,c), R(a,b), R(b, null)\}$, with $\Delta(D, D_1) = \{R(b, null)\}$, and $D_2 = \{P(a, null), R(a,b)\}$, with $\Delta(D, D_2) = \{P(b,c)\}$. Notice, for example, that $D_3 = \{P(a, null), P(b,c), R(a,b), R(b,d)\}$ is not a repair: since $\Delta(D, D_3) = \{R(b,d)\}$, we have $R(b, null) \sqsubseteq R(b,d)$ and, therefore $D_2 <_D D_3$. □

Example 5.6 Consider the UIC $\forall xy(P(x,y) \rightarrow T(x))$ and the RIC $\forall x(T(x) \rightarrow \exists y P(y, x))$, and the inconsistent database $D = \{P(a,b), P(null, a), T(c)\}$. In this case, we have a RIC-cyclic set of ICs. The four repairs are

i	D_i	$\Delta(D, D_i)$
1	$\{P(a,b), P(null, a), T(c), P(null, c), T(a)\}$	$\{T(a), P(null, c)\}$
2	$\{P(a,b), P(null, a), T(a)\}$	$\{T(a), T(c)\}$
3	$\{P(null, a), T(c), P(null, c)\}$	$\{P(a,b), P(null, c)\}$
4	$\{P(null, a)\}$	$\{P(a,b), T(c)\}$

Notice that, for example, the additional instance $D_5 = \{P(a, b),\ P(null, a),\ T(c),$ $P(c, a), T(c)\}$, with $\Delta(D, D_5) = \{T(a), P(c, a)\}$, satisfies the set of constraints IC, but is not a repair because $D_1 <_D D_5$. □

The previous example shows that with the new repair semantics we obtain a finite number of repairs (each with a finite extension) even when the set of ICs is cyclic. If we repaired the database by using the non-null constants in the infinite domain, with the repair semantics of [Arenas *et al.*, 1999], we would obtain an infinite number of repairs and infinitely many of them with infinite extension, as analyzed in [Calì *et al.*, 2003a].

In fact, it is possible to prove that under our repair semantics there will always exist a repair for a database D with respect to a set of ICs of the form 2.1. This follows from the fact that a database instance with no tuples always satisfies the ICs. Furthermore, the set of repairs is finite and each of them is finite in extension (i.e. each database relation is finite). This can be proved establishing first that the repairs are restricted to have constants in $adom(D) \cup const(IC) \cup \{null\}$, where $adom(D)$ is the active domain of the original instance D and $const(IC)$ is the set of constants that appear in the ICs. Since finitely many constants can appear in a repair, there is a finite set of database instances that are candidates to repairs, and each of them is finite.

The results in this section do not consider NNCs, but can be extended to them, as shown in Section 5.2.2.

Proposition 5.1 Given a database D and a set IC of RICs and UICs: (a) For every repair $D' \in Rep(D, IC)$, $adom(D') \subseteq adom(D) \cup const(IC) \cup \{null\}$. (b) The set $Rep(D, IC)$ of repairs is non-empty and finite; and every $D' \in Rep(D, IC)$ is finite.

Proof: (a) By contradiction, let us assume there is a repair $D' \in Rep(D, IC)$ with

an atom $R(\bar{a}, c)$ such that $c \notin (adom(D) \cup const(IC) \cup \{null\})$. Since $c \notin adom(D)$, $R(\bar{a}, c) \notin D$ and, therefore $R(\bar{a}, c) \in \Delta(D, D')$. $R(\bar{a}, c)$ could have been added to restore consistency with respect to a RIC or UIC. We have three cases: (i) Constant c corresponds to an existentially quantified attribute in the constraint. For database $D'' = (D' \cup \{R(\bar{a}, null)\}) \smallsetminus \{R(\bar{a}, c)\}$, we would have $D'' \models IC$ and $D'' <_D D'$, therefore D' would not be a repair. We have a contradiction. (ii) Constant c corresponds to a universally quantified attribute in an ICs. Since constraints have form (2.1), it implies that the atom(s) that created the inconsistency had c in at least one of its attributes. Here, we have two choices (ii.1) The atom that contains c was part of D, then $c \in adom(D)$, and this is a contradiction. (ii.2) The atom that contained c was added to solve an inconsistency, in which case we go back to the beginning of this argument. (iii) The constraint has a constant c, therefore $c \in const(IC)$, and we have reached a contradiction.

(b) First, let us prove that $Rep(D, IC)$ is non-empty. Consider a database instance D_0 compatible with Σ such that each predicate is empty. It is easy to check that $D_0 \models_N IC$ (note that there is always a predicate in the antecedent of the constraints). If there is no other instance D' over Σ such that $D' \models_N IC$ and $D' <_D D_0$, then D_0 is a repair. On the other hand, if there is D' such that $D' \models_N IC$ and $D' <_D D_0$, we can repeat the same argument; etc. Since the instances involved have all finite extensions for the database relations, it is not difficult to show that the partial order $\leq_D$ is well-founded, so there won't be an infinite decreasing chain.

Now let us prove that every $D' \in Rep(D, IC)$ is finite and the number of repairs is also finite. Part (A) shows that the active domain of the repairs is a subset of $(adom(D) \cup const(IC) \cup \{null\})$ (which is finite). Since the number of predicates is also finite, the set of databases that can be candidates to repairs are obtained from all the possible instantiations and combinations of predicates and atoms, and therefore

the number of repairs is finite and each of them is finite too. $\square$

Theorem 5.1 The problem of determining if a database D' is a repair of D with respect to a set IC consisting of ICs of the form (2.1) is *coNP*-complete. $\square$

In order to prove Theorem 5.1 we need to introduce some definitions and propositions that relate our repair semantics to the one introduced in [Chomicki and Marcinkowski, 2005a]. In [Chomicki and Marcinkowski, 2005a] repairs are obtained from the database by tuple deletion only, and the original database has no null values. We will refer to this type of repairs as *del-repairs*, and will denote the set of del-repairs by $Rep_{del}(D, IC)$.

Definition 5.4 [Chomicki and Marcinkowski, 2005a] Given a set IC of ICs and a database D, a database D' is a *del-repair* of D with respect to IC if D' is a maximal subset of D, such that $D' \models IC$. Let $Rep_{del}(D, IC)$ be the set of del-repairs of D with respect to IC. The *del-consistent* answer to a boolean query is *yes* if it is true in every del-repair of D with respect to IC. $\square$

Example 5.7 Give a database $D = \{P(a,b), R(b), S(a), P(c,e), R(e), T(b), T(e)\}$ and $IC = \{\forall xy((P(x,y) \wedge R(y)) \rightarrow S(x)), \forall x(T(x) \rightarrow R(x))\}$. The del-repairs of D with respect to IC are $D_1 = \{P(a,b), R(b), S(a), R(e), T(b), T(e)\}$ and $D_2 = \{P(a,b), R(b), S(a), P(c,e), T(b)\}$, with $(D \smallsetminus D_1) = \{P(c,e)\}$ and $(D \smallsetminus D_2) = \{R(e), T(e)\}$. $\square$

The next proposition shows that every del-repair is a repair.

Proposition 5.2 Given a database D without *null* and a set IC of UICs and RICs, if $D' \in Rep_{del}(D, IC)$, then $D' \in Rep(D, IC)$.

Proof: By contradiction, let us assume there exists a del-repair D_1 that is not a repair, i.e., $D_1 \in Rep_d(D, IC)$ and $D_1 \notin Rep(D, IC)$. Since D_1 is a del-repair, it holds:

1. $D_1 \models IC$.

2. $(D_1 \smallsetminus D) = \emptyset$. This is because D_1 is obtained only by tuple deletions.

3. There is no D_2 such that $D_2 \models IC$, $(D_2 - D) = \emptyset$ and $(D \smallsetminus D_2) \subsetneq (D - D_1)$.

4. D_1 has no *null*.

5. $D_1 \models_N IC$. This is because $\models_N$ coincides with $\models$ when there is no *null*.

Since $D_1 \notin Rep(D, IC)$ and $D_1 \models_N IC$, there must exist a database instance D_3 such that $D_3 \models_N IC$ and $D_3 <_D D_1$. By definition, $D_3 <_D D_1$ iff $D_3 \leq_D D_1$ and $D_1 \nleq_D D_3$. Then, since $D_3 \leq_D D_1$, the following holds:

- For every database atom $P(\bar{a}) \in \Delta(D, D_3)$, with $\bar{a} \in (\mathcal{U} \smallsetminus \{null\})$, it holds $P(\bar{a}) \in \Delta(D, D_1)$. Then, $P(\bar{a}) \in ((D \smallsetminus D_1) \cup (D_1 \smallsetminus D))$, but since $(D_1 \smallsetminus D) = \emptyset$, $P(\bar{a}) \in (D \smallsetminus D_1)$. Then, for all $P(\bar{a}) \in \Delta(D, D_3)$, $P(\bar{a}) \in (D \smallsetminus D_3)$. This implies that there are no tuple insertions without *null*.

- For every atom $Q(\bar{a}, \overline{null}) \in \Delta(D, D_3)$, with $\bar{a} \in (\mathcal{U} \smallsetminus \{null\})$, there exists $\bar{b} \in \mathcal{U}$ such that $Q(\bar{a}, \bar{b}) \in \Delta(D, D_1)$ and $Q(\bar{a}, \bar{b}) \notin \Delta(D, D_3)$. Since D has no *null* we know that $Q(\bar{a}, \overline{null}) \in (D_3 \smallsetminus D)$. Also, since $Q(\bar{a}, \bar{b}) \in \Delta(D, D_1)$, $Q(\bar{a}, \bar{b}) \in (D \smallsetminus D_1)$ and $Q(\bar{a}, \bar{b}) \in D$. Since $Q(\bar{a}, \bar{b}) \notin \Delta(D, D_3)$, $Q(\bar{a}, \bar{b}) \in D_3$. Let $D_4 = D_3 \smallsetminus \{Q(\bar{a}, \overline{null})\}$. It is easy to check that $D_4 \models_N IC$ and clearly $D_4 <_D D_3$. So, we can always construct an instance D_4, such that $\delta(D, D_4)$ does not contain *null* and $D_4 <_D D_1$.

Then, there exists an instance D_3 such that $D_3 \models_N IC$, $D_3 <_D D_1$ and $(D_3 \smallsetminus D) = \emptyset$. This implies that $(D \smallsetminus D_3) \subsetneq (D \smallsetminus D_1)$. Also, D_3 will have no *null* since D has no *null* and the repair was obtained only by tuple deletion. Given that $D_3 \models_N IC$ and D_3 has no *null*, we can conclude that $D_3 \models IC$. Since D_1 is also a del-repair, by condition in item 3 it holds that there should not exist such D_3. We have reached a contradiction. $\qquad\square$

Example 5.8 (example 5.7 continued) The repairs of D with respect to IC are the del-repairs D_1, D_2 plus $D_3 = \{P(a,b), R(b), S(a), P(c,e), R(e), T(b), T(e), S(c)\}$, where $S(c)$ was added to D. This is consistent with the result of Proposition 5.2. $\square$

The next proposition shows that every repair that is not a del-repair has at least one inconsistency that can be resolved by tuple insertion.

Proposition 5.3 Given a database D without null values and a set IC of UICs and RICs, if $D' \in Rep(D, IC)$ and $D' \smallsetminus D = \emptyset$, then $D' \in Rep_{del}(D, IC)$.

Proof: Since D' is a repair we know that D' is $<_D$-*minimal* and that $R \models_N IC$. In order to prove that D' is a del-repair, we need to show that it satisfies IC, that $(D' \smallsetminus D) = \emptyset$ and that there is no D'' such that $D' \subsetneq D''$, or in other words that $(D - D'') \subsetneq (D - D')$. We can prove the first by noting that since D has no *null* and $(D' \smallsetminus D) =$, D' has no *null* and therefore $R \models IC$. The second condition is an assumption of the problem. Now we only need to prove that there is no D'' such that $D'' \models IC$ and $(D - D'') \subsetneq (D - D')$. By contradiction we will assume that such D'' exists. Since there are no null values $D'' \models_N IC$ and $D'' <_D D'$. But this would imply that D' is not a repair. We have reached a contradiction. $\qquad\square$

Example 5.9 (example 5.8 continued) $D_1 \smallsetminus D = \emptyset$, $D_2 \smallsetminus D = \emptyset$ and $D_3 \smallsetminus D = \{S(c)\}$. As expected from Proposition 5.3, D_1 and D_2 are both del-repairs. $\qquad\square$

Proof of Theorem 5.1: The membership in *co-NP* can be proved by using the definition of repair. To prove that a database D' is not a repair of D with respect to IC it is enough to guess a consistent database D'' such that $D'' <_D D'$. The comparison $<_D$ can be done in polynomial time. Such witness, if exists, will have size polynomially bounded by the size of D' because of the definition of $<_D$.

Theorem 9 in [Chomicki and Marcinkowski, 2005a] proves that checking if a database is a del-repair of D with respect to a set of UICs and RICs IC is coNP-complete. In order to prove hardness they define a database D and a set IC of UICs and RICs, such that the empty set is a del-repair of D with respect to IC iff a propositional formula Φ is not satisfiable. From Propositions 5.2 and 5.3 we know that the empty set is a del-repair iff the empty set is a repair in our sense (note that the empty set can be a repair only if tuples where only deleted from the database). Therefore, we have proven that our problem is *coNP*-hard. □

We have shown that the problem of determining if a database instance is a repair of a database with respect to a set of ICs, is *coNP*-complete. Now we want to study the problem of retrieving consistent answers. The consistent answers to a query are the answers obtained from *all* the repairs of the database.

Definition 5.5 [Arenas *et al.*, 1999] Given a database D, a set of ICs IC, a query $Q(\bar{x})$, and a notion $\models^{\mathcal{Q}}$ of satisfaction in databases with *null*, a ground tuple $\bar{t}$ is a *consistent answer* to Q with respect to IC in D iff for every $D' \in Rep(D, IC)$, $D' \models^{\mathcal{Q}} Q[\bar{t}]$. If Q is a sentence (boolean query), then *yes* is a consistent answer iff $D' \models^{\mathcal{Q}} Q$ for every $D' \in Rep(D, IC)$. Otherwise, the consistent answer is *no*. □

In this formulation of CQA, $\models^{\mathcal{Q}}$ denotes the semantics of satisfaction of queries in a database with *null*. At this stage, we are not committing ourselves to any particular semantics for query answering. We will assume that we have such a notion that can

be applied to queries in databases with *null*.

Alternatives for this query semantics, among others, are those proposed in [Levene and Loizou, 1999a; Zaniolo, 1984], the SQL query answering semantics, and the null query answering semantics introduced in Chapter 4. We will also assume that $\models^{\mathcal{Q}}$ can be computed in polynomial time in data complexity for *safe* first-order queries [Gelder and Topor, 1987], and that it coincides with the classical first-order semantics for queries in databases without *null*. We will also assume in the following that queries are *safe*, a sufficient syntactic condition for domain independence [Gelder and Topor, 1987].

The decision problem for consistent query answering is

$$CQA(Q, IC) = \{(D, \bar{t}) \mid \bar{t} \text{ is a consistent answer to } Q(\bar{x}) \text{ with respect to } IC \text{ in } D\}.$$

Since we have Q and IC as parameters of the problem, we are interested in the data complexity of this problem, i.e., in terms of the size of the database [Abiteboul *et al.*, 1995]. It turns out that CQA for FOL queries is decidable, in contrast to what happens with the classic repair semantics [Arenas *et al.*, 1999], as established in [Calì *et al.*, 2003a]. The ideas behind the proof are as follows: (a) There is a finite number of database instances that are candidates to be repairs, because they use only the active domain of the original instance, *null*, and the constants in the ICs. (b) The satisfaction of ICs in the candidates can de decided by restriction to the active domain, because the ICs are domain independent. (c) Checking if $D_1 \leq_D D_2$ can be effectively decided. (d) The answers to safe first-order queries can be effectively computed.

Theorem 5.2 Consistent query answering for first-order queries with respect to sets of ICs of the form (2.1) is decidable. □

Proof: By Proposition 5.1, there are a finite number of database instances that are

candidates to be repairs of D with respect to a set of ICs IC. Let us denote this set with $CR(D, IC)$. We can obtain them by taking all the predicates in the schema with all the possible combinations of elements in $(adom(D) \cup const(IC) \cup \{null\})$. From this set of candidates to be repairs we need to check which of them satisfy the constraints, i.e., we need to compute $ConsisCR(D, IC) = \{D \mid D \in CR(D, IC) \text{ and } D \models_N IC\}$. In order to check if $D \models_N IC$, we need to check if $D \models IC^N$. For every $\psi \in IC$, ψ^N is domain independent, therefore the formula can be checked using the active domain. From $ConsisCR(D, IC)$, we can prune the database instances that are not $\leq_D$-*minimal* by doing a minimality test, and therefore obtaining $Rep(D, IC)$. For a safe query Q, we can compute the consistent answer by returning the tuples we get from *all* the different repairs in $Rep(D, IC)$. $\square$

The following theorem can be obtained by using a similar result in [Chomicki and Marcinkowski, 2005a] and the facts that our tuple deletion based repairs are exactly the repairs in [Chomicki and Marcinkowski, 2005a], and every repair in our sense that is not one of those contains at least one tuple insertion.

Theorem 5.3 Consistent query answering for first-order queries and sets of ICs of the form (2.1) is Π_2^P-complete.

Proof: Membership can be proved by using the definition consistent query answering. To prove that a query is not consistently true, we need to find a database repair in which the query is false. Checking if a database is a repair is in *co-NP*.

In Theorem 12 in [Chomicki and Marcinkowski, 2005a] it is proved that the problem of checking if a tuple is a del-consistent answer is Π_2^P-complete. In order to prove hardness Chomicki and Marcinkowski define a database D_1, a set IC_1 with one functional dependency, and a RIC such that $R(a, a, \psi_1, a)$ is a del-consistent answer iff the following quantified boolean formula is true: $\forall \bar{p} \exists \bar{q} (\psi_1 \wedge \cdots \wedge \psi_m)$, where ψ_i are

clauses. From Propositions 5.2 and 5.3 we know that, given a database D without *null*, all del-repairs of D with respect to IC are repairs of D with respect to IC, and that for every repair D' that is not a del-repair, $(D' \smallsetminus D) \neq \emptyset$. For IC_1, the constraints from the reduction, the only way in which an inconsistency can be solved by inserting a tuple is the RIC, since the functional dependency can be solved only by tuple deletions. Therefore, we are sure that if a tuple is inserted it will contain *null* (if a value different from *null* is inserted in the existential variable, the new database would not be $<_D$-*minimal*). Then, we can conclude that if $D' \in Rep(D_1, IC_1)$, then only one of the following holds $D' \in Rep_{del}(D_1, IC_1)$ or D_1 has *null* in it. Let N be boolean query such that the answer to it is *Yes* iff *null* is in the database. It is easy to see that $R(a, a, \psi_1, a)$ is a del-consistent answer of D_1 with respect to IC_1 iff $R(a, a, \psi_1, a) \vee N$ is a consistent answer of D_1 with respect to IC_1. Therefore, our problem is Π_2^P-hard. □

Note that hardness can be obtained already for boolean queries.

5.2   Repair Logic Programs

Repairs of relational databases can be specified as stable models of disjunctive logic programs. In [Barceló and Bertossi, 2002; Barceló; and Bertossi, 2003; Barceló *et al.*, 2003; Bravo and Bertossi, 2004; Caniupan and Bertossi, 2005] such programs were presented, but they were based on classic IC satisfaction, that differs from the one introduced in Section 4.1. These different semantics for satisfaction of ICs produces as a consequence a different repair semantics. Furthermore, the results in [Barceló and Bertossi, 2002; Barceló; and Bertossi, 2003] consider a different repair semantics for RICs, where they are repaired using arbitrary values from the domain.

The repair programs that we will present now implement the new repair semantics for a set of RIC-acyclic constraints. The program $\Pi(D, IC)$ from Definition 5.1 needs

to be adjusted to deal with the new repair semantics for RICs and with the semantics of IC satisfaction in the presence of *null*.

Definition 5.6 Given a database D and a set IC of UICs of the form (2.2) and of RICs of the form (2.3), the *repair program* $\Pi(D, IC)$ is a disjunctive logic program formed by the following set of rules:

1. $dom(x)$ for every constant x in database D except *null*.

2. The fact $P\_(\bar{a}, \mathbf{t_d})$ for every atom $P(\bar{a}) \in D$.

3. For every UIC ψ of the form (2.2), the rules:

$$\bigvee_{i=1}^{n} P_i\_(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} Q_j\_(\bar{y}_j, \mathbf{t_a}) \longleftarrow \bigwedge_{i=1}^{n} P_i\_(\bar{x}_i, \mathbf{t^\star}), \bigwedge_{j=1}^{m} Q_j\_(\bar{y}_j, \mathbf{f^\star}),$$

$$\bigwedge_{x_l \in \mathcal{A}(\psi)} dom(x_l), \bar{\varphi}.$$

where $\mathcal{A}(\psi)$ 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 RIC ψ of the form (2.3), the rules:[3]

$$P\_(\bar{x}, \mathbf{f_a}) \vee Q\_(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow P\_(\bar{x}, \mathbf{t^\star}), \ not \ aux_\psi(\bar{x}'), \ dom(\bar{x}').$$

And for every $y_i \in \bar{y}$:

$$aux_\psi(\bar{x}') \leftarrow Q\_(\bar{x}', \bar{y}, \mathbf{t^\star}), \ not \ Q\_(\bar{x}', \bar{y}, \mathbf{f_a}), \ dom(\bar{x}'), \ dom(y_i).$$

$$aux_\psi(\bar{x}') \leftarrow Q\_(\bar{x}', \overline{null}, \mathbf{t_d}), \ not \ Q\_(\bar{x}', \overline{null}, \mathbf{f_a}), \ dom(\bar{x}').$$

5. For every predicate $P \in \mathcal{R}$, $\Pi(D, IC)$ contains the rules:

$$P\_(\bar{x}, \mathbf{t^\star}) \leftarrow P\_(\bar{x}, \mathbf{t_d}). \qquad P\_(\bar{x}, \mathbf{t^\star}) \leftarrow P\_(\bar{x}, \mathbf{t_a}).$$

$$P\_(\bar{x}, \mathbf{f^\star}) \leftarrow P(\bar{x}, \mathbf{f_a}). \qquad P\_(\bar{x}, \mathbf{f^\star}) \leftarrow dom(\bar{x}), \ not \ P\_(\bar{x}, \mathbf{t_d}).$$

6. For every predicate $P \in \mathcal{R}$, $\Pi(D, IC)$ contains ~~the rules~~

$$P\_(\bar{x}, \mathbf{f^{\star\star}}) \leftarrow P\_(\bar{x}, \mathbf{f_a}). \qquad P\_(\bar{x}, \mathbf{f^{\star\star}}) \leftarrow dom(\bar{x}), \ not \ P\_(\bar{x}, \mathbf{t_d}), \ not \ P\_(\bar{x}, \mathbf{t_a}).$$

---

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

$$P_-(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P_-(\bar{x}, \mathbf{t_a}). \qquad P_-(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P_-(\bar{x}, \mathbf{t_d}), \ not \ P_-(\bar{x}, \mathbf{f_a}).$$

7. For every predicate $P \in \mathcal{R}$, $\Pi(D, IC)$ contains the rules:

$$\leftarrow P_-(\bar{x}, \mathbf{t_a}) \ , P_-(\bar{x}, \mathbf{f_a}). \hspace{4cm} \square$$

Facts in 1. are the elements of the domain and facts in 2. are the elements in the database. Rules in 3. enforce, as before, the satisfaction of UICs. Rules in 4. enforce the satisfaction of a RIC. If $P_-(\bar{a}, \mathbf{t}^{\star})$ is true and $aux(\bar{a}')$ is false, i.e., there is no $\bar{y}$ such that $Q(\bar{a}', \bar{y})$ is true or was made true by the repair program, then there are two alternative ways to restore consistency: delete $P(\bar{a})$ or add $Q(\bar{a}', n\bar{u}ll)$ to the database. Since the satisfaction of UICs and RICs needs to be checked only if none of the relevant attributes of the antecedent are *null*, we use $dom(x)$ in rule 3. and in the rules in 4. so that the constraint is checked only if x is different from *null* (since $dom(null)$ is always *false*). Notice that rules 4. are implicitly based on the fact that the relevant attributes for a RIC of the form (2.3) are $\mathcal{A} = \{x \mid x \in \bar{x}'\}$. Rules in 5. capture the atoms that are part of the inconsistent database or that become true in the repair process; and rules 6. those that become true in the repairs. Rule 7. enforces, by discarding models, that no atom can be made both true and false in a repair. Note that in the repair program, *null* is treated as any other constant.

For a stable model $\mathcal{M}$ of the program, $D_\mathcal{M}$ denotes the database obtained from it by keeping only the tuples with annotation constant $\mathbf{t}^{\star\star}$.

Definition 5.7 Let $\mathcal{M}$ be a stable model of program $\Pi(D, IC)$. The database associated to $\mathcal{M}$ is $D_\mathcal{M} = \{P(\bar{a}) \mid P_-(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}\}$. $\hspace{2cm} \square$

Example 5.10 Consider the database instance $\{P(\bar{a})\}$ and the ICs: $\{\forall xy(Q(x, y) \rightarrow R(x, y)), \forall x(P(x) \rightarrow \exists y Q(x, y))\}$. The program $\Pi(D, IC)$ is:

1. $dom(a)$.

2. $P_-(a, \mathbf{t_d})$.

3. $Q_-(x, y, \mathbf{f_a}) \vee R_-(x, y, \mathbf{t_a}) \leftarrow Q_-(x, y, \mathbf{t^\star}), R_-(x, y, \mathbf{f^\star}), dom(x), dom(y)$.

4. $P_-(x, \mathbf{f_a}) \vee Q_-(x, null, \mathbf{t_a}) \leftarrow P_-(x, \mathbf{t^\star}), \ not \ aux_1(x), dom(x)$.

   $aux_1(x) \leftarrow Q_-(x, y, \mathbf{t^\star}), \ not \ Q_-(x, y, \mathbf{f_a}), dom(x), dom(y)$.

   $aux_1(x) \leftarrow Q_-(x, null, \mathbf{t_d}), \ not \ Q_-(x, null, \mathbf{f_a}), dom(x)$.

5. $P_-(x, \mathbf{f^\star}) \leftarrow dom(x), \ not \ P_-(x, \mathbf{t_d})$.

   $P_-(x, \mathbf{f^\star}) \leftarrow P_-(x, \mathbf{f_a})$.

   $P_-(x, \mathbf{t^\star}) \leftarrow P_-(x, \mathbf{t_a})$.

   $P_-(x, \mathbf{t^\star}) \leftarrow P_-(x, \mathbf{t_d})$.

   $Q_-(x, y, \mathbf{f^\star}) \leftarrow dom(x), dom(y), \ not \ Q_-(x, y, \mathbf{t_d})$.

   $Q_-(x, y, \mathbf{f^\star}) \leftarrow Q_-(x, y, \mathbf{f_a})$.

   $Q_-(x, y, \mathbf{t^\star}) \leftarrow Q_-(x, y, \mathbf{t_a})$.

   $Q_-(x, y, \mathbf{t^\star}) \leftarrow Q_-(x, y, \mathbf{t_d})$.

   $R_-(x, y, \mathbf{f^\star}) \leftarrow dom(x), dom(y), \ not \ R_-(x, y, \mathbf{t_d})$.

   $R_-(x, y, \mathbf{f^\star}) \leftarrow R_-(x, y, \mathbf{f_a})$.

   $R_-(x, y, \mathbf{t^\star}) \leftarrow R_-(x, y, \mathbf{t_a})$.

   $R_-(x, y, \mathbf{t^\star}) \leftarrow R_-(x, y, \mathbf{t_d})$.

6. $P_-(x, \mathbf{t^{\star\star}}) \leftarrow P_-(x, \mathbf{t_a})$.

   $P_-(x, \mathbf{t^{\star\star}}) \leftarrow P_-(x, \mathbf{t_d}), \ not \ P_-(x, \mathbf{f_a})$.

   $P_-(x, \mathbf{f^{\star\star}}) \leftarrow P_-(x, \mathbf{f_a})$.

   $P_-(x, \mathbf{f^{\star\star}}) \leftarrow dom(x), \ not \ P_-(x, \mathbf{t_d}), \ not \ P_-(x, \mathbf{t_a})$.

   $Q_-(x, y, \mathbf{t^{\star\star}}) \leftarrow Q_-(x, y, \mathbf{t_a})$.

   $Q_-(x, y, \mathbf{t^{\star\star}}) \leftarrow Q_-(x, y, \mathbf{t_d}), \ not \ Q_-(x, y, \mathbf{f_a})$.

   $Q_-(x, y, \mathbf{f^{\star\star}}) \leftarrow Q_-(x, y, \mathbf{f_a})$.

   $Q_-(x, y, \mathbf{f^{\star\star}}) \leftarrow dom(x), dom(y), \ not \ Q_-(x, y, \mathbf{t_d}), \ not \ Q_-(x, y, \mathbf{t_a})$.

$R_-(x, y, \mathbf{t}^{\star\star}) \leftarrow R_-(x, y, \mathbf{t_a})$.

$R_-(x, y, \mathbf{t}^{\star\star}) \leftarrow R_-(x, y, \mathbf{t_d}), \ not \ Q_-(x, y, \mathbf{f_a})$.

$R_-(x, y, \mathbf{f}^{\star\star}) \leftarrow R_-(x, y, \mathbf{f_a})$.

$R_-(x, y, \mathbf{f}^{\star\star}) \leftarrow dom(x), dom(y), \ not \ R_-(x, y, \mathbf{t_d}), \ not \ R_-(x, y, \mathbf{t_a})$.

7. $\leftarrow P_-(x, \mathbf{t_a}), P_-(x, \mathbf{f_a})$.

$\leftarrow Q_-(x, y, \mathbf{t_a}), Q_-(x, y, \mathbf{f_a})$.

$\leftarrow R_-(x, y, \mathbf{t_a}), R_-(x, y, \mathbf{f_a})$.

The stable models of the program are:

$\mathcal{M}_1 = \{dom(a), P_-(a, \mathbf{t_d}), P_-(a, \mathbf{t}^\star), P_-(a, \mathbf{f}^\star), P_-(a, \mathbf{f}^{\star\star}), P_-(a, \mathbf{f_a}), Q_-(a, a, \mathbf{f}^\star),$

$\qquad R_-(a, a, \mathbf{f}^\star), Q_-(a, a, \mathbf{f}^{\star\star}), R_-(a, a, \mathbf{f}^{\star\star})\}$

$\mathcal{M}_2 = \{dom(a), \ P_-(a, \mathbf{t_d}), \ P_-(a, \mathbf{t}^\star), \ \underline{P_-(a, \mathbf{t}^{\star\star})}, \ Q_-(a, null, \mathbf{t_a}), \ Q_-(a, a, \mathbf{f}^\star),$

$\qquad R_-(a, a, \mathbf{f}^\star), Q_-(a, a, \mathbf{f}^{\star\star}), R_-(a, a, \mathbf{f}^{\star\star}), \underline{Q_-(a, null, \mathbf{t}^{\star\star})}\}$

The databases associated with these models are $D_{\mathcal{M}_1} = \emptyset$ and $D_{\mathcal{M}_2} = \{P(a), \ Q(a, null)\}$, and they correspond to the repairs.

If the fact $Q(a, null)$ is added to the instance, the fact $Q_-(a, null, \mathbf{t_d})$ becomes a part of the program. Since the database is consistent in this case, the program would have only one stable model, $\mathcal{M}_1$, corresponding to $\{P(a), Q(a, null)\}$, the original database:

$\mathcal{M}_1 = \{dom(a), \ P_-(a, \mathbf{t_d}), \ P_-(a, \mathbf{t}^\star), \ Q_-(a, null, \mathbf{t_d}), \ \underline{P_-(a, \mathbf{t}^{\star\star})}, \ Q_-(a, a, \mathbf{f}^\star),$

$\qquad R_-(a, a, \mathbf{f}^\star), Q_-(a, a, \mathbf{f}^{\star\star}), R_-(a, a, \mathbf{f}^{\star\star}), \underline{Q_-(a, null, \mathbf{t}^{\star\star})}\}$.

$\square$

Example 5.11 (example 5.3 continued) The repair program $\Pi(D, IC)$:

1. $dom(21), \quad dom(C15). \quad dom(34), \quad dom(C18). \quad dom(21),$

$\quad dom(Ann). \quad dom(45), \quad dom(Paul).$

2. $Reg\_(21, C15, \mathbf{t_d})$. $Reg\_(34, C18, \mathbf{t_d})$.

   $Student\_(21, Ann, \mathbf{t_d})$. $Student\_(45, Paul, \mathbf{t_d})$.

3. There is no UIC.

4. $Reg\_(x, y, \mathbf{f_a}) \vee Student\_(x, null, \mathbf{t_a}) \leftarrow Reg\_(x, y, \mathbf{t^\star}), not\ aux(x),\ x \neq null$.

   $aux(x) \leftarrow Student\_(x, y, \mathbf{t^\star}),\ not\ Student\_(x, y, \mathbf{f_a}),\ x \neq null,\ y \neq null$.

   $aux(x) \leftarrow Student\_(x, null, \mathbf{t_d}),\ not\ Student\_(x, null, \mathbf{f_a}),\ x \neq null$.

5. $Reg\_(x, y, \mathbf{t^\star}) \leftarrow Reg\_(x, y, \mathbf{t_a})$.

   $Reg\_(x, y, \mathbf{t^\star}) \leftarrow Reg\_(x, y, \mathbf{t_d})$.

   $Reg\_(x, y, \mathbf{f^\star}) \leftarrow Reg\_(x, y, \mathbf{f_a})$.   (Similarly for *Student*)

   $Reg\_(x, y, \mathbf{f^\star}) \leftarrow dom(x), dom(y),\ not\ Reg\_(x, y, \mathbf{t_d})$.

6. $Reg\_(x, y, \mathbf{t^{\star\star}}) \leftarrow Reg\_(x, y, \mathbf{t_a})$.

   $Reg\_(x, y, \mathbf{t^{\star\star}}) \leftarrow Reg\_(x, y, \mathbf{t_d}),\ not\ Reg\_(x, y, \mathbf{f_a})$.

   $Reg\_(x, y, \mathbf{f^{\star\star}}) \leftarrow Reg\_(x, y, \mathbf{f_a})$.   (Similarly for *Student*)

   $Reg\_(x, y, \mathbf{f^{\star\star}}) \leftarrow dom(x), dom(y),\ not\ Reg\_(x, y, \mathbf{t_d})$,

   $\qquad\qquad\qquad\quad not\ Reg\_(x, y, \mathbf{t_a})$.

7. $\leftarrow Reg\_(x, y, \mathbf{t_a}), Reg\_(x, y, \mathbf{f_a})$.

   $\leftarrow Student\_(x, y, \mathbf{t_a}), Student\_(x, y, \mathbf{f_a})$.

The program has two stable models:[4]

$\mathcal{M}_1 = \{ Reg\_(21, C15, \mathbf{t^\star}),\ Reg\_(34, C18,\ \mathbf{t^\star}),\ Student\_(21, Ann, \mathbf{t^\star}),\ Student\_(45,$

$\qquad\quad Paul,\ \mathbf{t^\star}),\ Reg\_(34, C18,\ \mathbf{f_a}),\ \underline{Reg\_(21, C15, \mathbf{t^{\star\star}})},\ \underline{Student\_(21, Ann, \mathbf{t^{\star\star}})},$

$\qquad\quad \underline{Student\_(45, Paul, \mathbf{t^{\star\star}})}\}$.

---

[4]Facts and atoms annotated with $\mathbf{f^\star}$ and $\mathbf{f^{\star\star}}$ are omitted in all the model.

$$\mathcal{M}_2 = \{Reg\_(21, C15, \mathbf{t}^\star),\ Reg\_(34, C18,\ \mathbf{t}^\star),\ Student\_(21, Ann, \mathbf{t}^\star),\ Student\_(45,$$
$$Paul,\ \mathbf{t}^\star),\ Student\_(34,\ null, \mathbf{t_a}),\ \underline{Reg\_(21, C15, \mathbf{t}^{\star\star})},\ \underline{Student\_(21, Ann, \mathbf{t}^{\star\star})},$$
$$\underline{Student\_(45, Paul, \mathbf{t}^{\star\star})}\ \underline{Student\_(34, null, \mathbf{t}^{\star\star})}\}.$$

The databases associated to these models correspond exactly to the repairs:

$$D_{\mathcal{M}_1} = \{Reg(21, C15),\ Student(21, Ann),\ Student(45, Paul)\}.$$

$$D_{\mathcal{M}_2} = \{Reg(21, C15),\ Reg(34,\ C18),\ Student(21,\ Ann),\ Student(45,\ Paul),\ Student$$
$$(34,\ null)\}. \hspace{1cm} \square$$

Under this new repair semantics it holds that:

Theorem 5.4 Given a database D and a RIC-acyclic set of UICs and RICs, if $\mathcal{M}$ is a stable model of $\Pi(D, IC)$, then $D_{\mathcal{M}}$ is a repair of D with respect to IC. Furthermore, the repairs obtained in this way are all the repairs of D. $\hspace{1cm} \square$

In order to prove this theorem we need to introduce some lemmas, propositions and definitions.

Lemma 5.1 Given a database D and a RIC-acyclic set of UICs and RICs, if $\mathcal{M}$ is a stable model of $\Pi(D, IC)$, i.e., a minimal model of $\Pi(D,\ IC)^{\mathcal{M}}$, then exactly one of the following cases holds:

1. $P(\bar{a}, \mathbf{t_d})$, $P(\bar{a}, \mathbf{t}^\star)$ and $P(\bar{a}, \mathbf{t}^{\star\star})$ belong to $\mathcal{M}$, and no other $P\_(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.

2. $P(\bar{a}, \mathbf{t_d})$, $P(\bar{a}, \mathbf{t}^\star)$ and $P(\bar{a}, \mathbf{f_a})$ belong to $\mathcal{M}$, and no other $P\_(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.

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

4. $P(\bar{a}) \notin \mathcal{M}$ and no $P(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$.

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

- $P(\bar{a}, \mathbf{t_d}) \in \mathcal{M}$. Then, from rule 5, $P(\bar{a}, \mathbf{t}^\star) \in \mathcal{M}$. Two cases are possible now: $P(\bar{a}, \mathbf{f_a}) \notin \mathcal{M}$ or $P(\bar{a}, \mathbf{f_a}) \in \mathcal{M}$. For the first case, since $\mathcal{M}$ is minimal, $P(\bar{a}, \mathbf{t_a}) \notin \mathcal{M}$) and $P(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}$. For the second case, because of rule 7, $P(\bar{a}, \mathbf{t_a}) \notin \mathcal{M}$. These cases cover the first two items in the lemma.

- $P(\bar{a}, \mathbf{t_d}) \notin \mathcal{M}$. Two cases are possible now: $P(\bar{a}, \mathbf{t_a}) \in \mathcal{M}$ or $P(\bar{a}, \mathbf{t_a}) \notin \mathcal{M}$. For the first one we also have $P(\bar{a}, \mathbf{t}^{\star\star})$, $P(\bar{a}, \mathbf{t}^\star) \in \mathcal{M}$ because of rules 5 and 6 and $P(\bar{a}, \mathbf{f_a}) \notin \mathcal{M}$ because of rule 7. For the second one, $P(\bar{a}, \mathbf{t}^\star) \notin \mathcal{M}$ (since $\mathcal{M}$ is minimal), $P(\bar{a}, \mathbf{f_a}) \notin \mathcal{M}$ (because $P(\bar{a}, \mathbf{t}^\star) \notin \mathcal{M}$ and $\mathcal{M}$ is minimal). These cases cover the last two items in the lemma. $\qquad\square$

From two database instances we can define a structure.

Definition 5.8 For two database instances D_1 and D_2 over the same schema and domain and a set of ICs IC, $\mathcal{M}^\star_{IC}(D_1, D_2)$ is the Herbrand structure $\langle D, I_\mathcal{P}, I_\mathcal{B} \rangle$, where $\mathcal{U}$ is the domain of the database[5] 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 $P(\bar{a}, \mathbf{t_d})$, $P(\bar{a}, \mathbf{t}^\star)$ 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 $P(\bar{a}, \mathbf{t_d})$, $P(\bar{a}, \mathbf{t}^\star)$ and $P(\bar{a}, \mathbf{f_a}) \in I_P$.

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 all annotated constants v.

4. If $P(\bar{a}) \notin D_1$ and $P(\bar{a}) \in D_2$, then $P(\bar{a}, \mathbf{t_a})$, $P(\bar{a}, \mathbf{t}^\star)$ and $P(\bar{a}, \mathbf{t}^{\star\star}) \in I_\mathcal{P}$.

[5]Strictly speaking, the domain $\mathcal{U}$ now also contains the annotations values.

5. For every RIC $\psi \in IC$ of the form $\forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{y} Q(\bar{x}', \bar{y}))$. If $P(\bar{a}, \mathbf{t}^{\star\star}) \in I_{\mathcal{P}}$, for $\bar{a} \neq null$ and there exists a $\bar{b}$ with at least one $b \in \bar{b}$ such that $b \neq null$ and $Q(\bar{a}', \bar{b}, \mathbf{t}^{\star\star}) \in I_{\mathcal{P}}$, then $aux_{\psi}(\bar{a}') \in I_{\mathcal{P}}$.

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. $\qquad \square$

Notice that the database associated to $\mathcal{M}_{IC}^{\star}(D_1, D_2)$ corresponds exactly to D_2, i.e., $D_{\mathcal{M}_{IC}^{\star}(D_1, D_2)} = D_2$.

Lemma 5.2 Given a database D and a set of UICs and RICs, if $D' \models_{N} IC$, then there is a model $\mathcal{M}$ of the program $(\Pi(D, IC))^{\mathcal{M}}$ such that $D_{\mathcal{M}} = D'$. Furthermore, this model is $\mathcal{M}_{IC}^{\star}(D, D')$.

Proof: As $\mathcal{M}_{IC}^{\star}(D, D') = D'$, we only need to show $\mathcal{M}_{IC}^{\star}(D, D')$ satisfies all the rules of $(\Pi(D, IC))^{\mathcal{M}}$. It is clear that, by construction, rules 2, 5 and 6 are satisfied by $\mathcal{M}_{IC}^{\star}(D, D')$. For every UIC in IC there is a set of rules of the form 3. If the body of the rule is satisfied, then the atoms $P_i(\bar{a}_i, \mathbf{t}^{\star}) \in \mathcal{M}_{IC}^{\star}(D, D')$ and $Q_i(\bar{b}_i, \mathbf{f_a}) \in \mathcal{M}_{IC}^{\star}(D, D')$ or $Q_i(\bar{b}_i) \notin \mathcal{M}_{IC}^{\star}(D, D')$. 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 $\mathcal{M}_{IC}^{\star}(D, D')$, at least one of $P_i(\bar{a}_i, \mathbf{f_a})$ or $Q_i(\bar{b}_i, \mathbf{t_a})$ is in $\mathcal{M}_{IC}^{\star}(D, D')$. Therefore, the head of the rule is also satisfied and the whole rule is satisfied. For every RIC in IC, there is a set of rules of the form 4. By construction of $\mathcal{M}_{IC}^{\star}(D, D')$, the rules that define $aux_{\psi}(\bar{x})$ for all $\psi \in IC$ are satisfied. If the body of the first rule in 4 is true in $\mathcal{M}_{IC}^{\star}(D, D')$, it means that the integrity constraint is not satisfied in the database or at some point of the repair process. Since the constraint is satisfied by D', the satisfaction had to be restored by adding $Q(\bar{x}, \overline{null})$ or by deleting $P(\bar{x})$. This implies that $Q(\bar{x}, \overline{null}, \mathbf{t_a}) \in \mathcal{M}_{IC}^{\star}(D, D')$ or $P(\bar{x}, \mathbf{f_a}) \in \mathcal{M}_{IC}^{\star}(D, D')$, and therefore

that the first (or second) rule is satisfied. Then, by construction of $\mathcal{M}_{IC}^\star(D, D')$, $P_-(\bar{a}, \cancel{null}, \mathbf{f_a}) \in \mathcal{M}_{IC}^\star(D, D')$, and the head of the rule is also satisfied. $\qquad\square$

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

Lemma 5.3 Given a database D and a set of UICs and RICs, if $\mathcal{M}$ is a stable model of the program $\Pi(D, IC)$, then $D_\mathcal{M} \models_N IC$.

Proof: We want to show that $D_\mathcal{M} \models_N \psi$, for every constraint $\psi \in IC$. There are ~~three~~ cases to consider:

- IC ψ is a UIC. Since $\mathcal{M}$ is a model of $(\Pi(D, IC))^\mathcal{M}$, $\mathcal{M}$ satisfies rules 3 of $\Pi(D, IC)$. Then, at least one of the following cases holds:

  - $\mathcal{M} \models_N P_i(\bar{a}, \mathbf{f_a})$. Then, $\mathcal{M} \not\models_N P_i(\bar{a}, \mathbf{t}^{\star\star})$ and $P(\bar{a}) \notin D_\mathcal{M}$ (by Lemma 5.1). Hence, $D_\mathcal{M} \models_N \neg P_i(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_\mathcal{M} \models_N \bigwedge_{i=1}^n P_i(\bar{x}_i) \rightarrow \bigvee_{j=1}^m Q_j(\bar{y}_j) \vee \varphi$ holds.

  - $\mathcal{M} \models_N Q_j(\bar{a}, \mathbf{t_a})$. It is symmetric to the previous one.

  - It is not true that $\mathcal{M} \models_N \bar{\varphi}$. Then $\mathcal{M} \models_N \varphi$. Hence, φ is true, and $D_\mathcal{M} \models_N \bigwedge_{i=1}^n P_i(\bar{x}_i) \rightarrow \bigvee_{j=1}^m Q_j(\bar{y}_j) \vee \varphi$ holds.

  - $\mathcal{M} \not\models_N P_i(\bar{a}, \mathbf{t}^\star)$. Given that $\mathcal{M}$ is minimal, just the last item in Lemma 5.1 holds. This means $\mathcal{M} \not\models_N P_i(\bar{a}, \mathbf{t}^{\star\star})$, $P_i(\bar{a}) \notin D_\mathcal{M}$ and $D_\mathcal{M} \models_N \neg P_i(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_\mathcal{M} \models_N \bigwedge_{i=1}^n P_i(\bar{x}_i) \rightarrow \bigvee_{j=1}^m Q_j(\bar{y}_j) \vee \varphi$ holds.

  - $\mathcal{M} \not\models_N Q_j(\bar{a}, \mathbf{f_a})$ or $\mathcal{M} \models_N Q_j(\bar{a}, \mathbf{t_d})$. Given that $\mathcal{M}$ is minimal, just the first item in Lemma 5.1 holds. Then, $\mathcal{M} \models_N Q_j(\bar{a}, \mathbf{t}^{\star\star})$, $Q_j(\bar{a}) \in D_\mathcal{M}$ and $D_\mathcal{M} \models_N Q_j(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_\mathcal{M} \models_N \bigwedge_{i=1}^n P_i(\bar{x}_i) \rightarrow \bigvee_{j=1}^m Q_j(\bar{y}_j) \vee \varphi$ holds.

- Formula ψ is a RIC. Since $\mathcal{M}$ is a model of $(\Pi(D, IC))^{\mathcal{M}}$, $\mathcal{M}$ satisfies rules 4 of $\Pi(D, IC)$. Then, at least one of the following cases holds:

  - $\mathcal{M} \models_N P(\bar{a}, \mathbf{f_a})$. Then, $\mathcal{M} \not\models_N P_i(\bar{a}, \mathbf{t}^{\star\star})$ and $P(\bar{a}) \notin D_{\mathcal{M}}$ (by Lemma 5.1). Hence, $D_{\mathcal{M}} \models_N \neg P_i(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_{\mathcal{M}} \models_N (P(\bar{x}) \rightarrow Q(\bar{x}', y))$ holds.

  - $\mathcal{M} \models_N Q(\bar{a}', \overline{null}, \mathbf{t_a})$. It is symmetric to the previous one.

  - $\mathcal{M} \not\models_N P(\bar{a}, \mathbf{t}^{\star})$. Given that $\mathcal{M}$ is minimal, just the last item in Lemma 5.1 holds. This means $\mathcal{M} \not\models_N P(\bar{a}, \mathbf{t}^{\star\star})$, $P(\bar{a}) \notin D_{\mathcal{M}}$ and $D_{\mathcal{M}} \models_N \neg P(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_{\mathcal{M}} \models_N (P(\bar{x}) \rightarrow Q(\bar{x}', y))$ holds.

  - $\mathcal{M} \models_N aux_\psi(\bar{a}')$. This means that $P(\bar{a}, \mathbf{t}^{\star}) \in \mathcal{M}$ and that there exists $\bar{b} \neq \overline{null}$ such that $Q(\bar{a}', \bar{b}, \mathbf{t}^{\star}) \in \mathcal{M}$, $Q(\bar{a}', \bar{b}, \mathbf{f_a}) \notin \mathcal{M}$, and therefore that $P(\bar{a}) \in D_{\mathcal{M}}$ and $Q(\bar{a}, \bar{b}) \in D_{\mathcal{M}}$. Then, the constraint is satisfied. $\qquad\square$

Lemma 5.4 Consider two database instances D and D' over the same schema and domain. If $\mathcal{M}$ is a minimal model of $(\Pi(D, IC))^{\mathcal{M}^\star(D,D')}$ such that $\mathcal{M} \subsetneq \mathcal{M}^\star(D, D')$, then there exists $\mathcal{M}'$ that is a minimal model of $(\Pi(D, IC))^{\mathcal{M}'}$ with $D_{\mathcal{M}'} <_D D'$.

Proof: Since $\mathcal{M}$ is a minimal model of $(\Pi(D, IC))^{\mathcal{M}^\star_{IC}(D,D')}$, $P(\bar{a}, \mathbf{t_d}) \in \mathcal{M}$ iff $P(\bar{a}) \in D$. By definition of $\mathcal{M}^\star_{IC}(D, D')$ and $\mathcal{M} \subsetneq \mathcal{M}^\star(D, D')$, the only two ways that both models can differ is that, for some $P(\bar{a}) \in D$, $\{P(\bar{a}, \mathbf{f_a})\} \subseteq \mathcal{M}^\star(D, D')$ and $P(\bar{a})$ nor $P(\bar{a}, \mathbf{f_a})$ belong to $\mathcal{M}$, or for some $P(\bar{a}) \notin D$, $\{P(\bar{a}, \mathbf{t_a}), P(\bar{a}, \mathbf{t}^{\star}), P(\bar{a}, \mathbf{t}^{\star\star})\} \subseteq \mathcal{M}^\star_{IC}(D, D')$ and $P(\bar{a})$, $P(\bar{a}, \mathbf{t_a})$, $P(\bar{a}, \mathbf{t}^{\star})$ nor $P(\bar{a}, \mathbf{t}^{\star\star})$ belong to $\mathcal{M}$. Now, if we use the interpretation rules over $\mathcal{M}$, we will construct a model $\mathcal{M}'$ that is a minimal model of $(\Pi(D, IC))^{\mathcal{M}'}$. From $\mathcal{M}$, the model $\mathcal{M}'$ is constructed as follows:

- If $P(\bar{a}, \mathbf{t_d}) \in \mathcal{M}$ and $P(\bar{a}, \mathbf{f_a}) \notin \mathcal{M}$, then $P(\bar{a}, \mathbf{t_d}), P(\bar{a}, \mathbf{t}^{\star})$ and $P(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}'$.

- If $P(\bar{a}, \mathbf{t_d}) \in \mathcal{M}$ and $P(\bar{a}, \mathbf{f_a}) \in \mathcal{M}$, then $P(\bar{a}, \mathbf{t_d}), P(\bar{a}, \mathbf{t}^\star)$ and $P(\bar{a}, \mathbf{f_a}) \in \mathcal{M}'$.

- If $P(\bar{a}, \mathbf{t_d}) \notin \mathcal{M}$ and $P(\bar{a}, \mathbf{t_a}) \notin \mathcal{M}$, then nothing is added to $\mathcal{M}'$.

- If $P(\bar{a}, \mathbf{t_d}) \notin \mathcal{M}$ and $P(\bar{a}, \mathbf{t_a}) \in \mathcal{M}$, then $P(\bar{a}, \mathbf{t_a}), P(\bar{a}, \mathbf{t}^\star)$ and $P(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}'$.

It is clear that $\mathcal{M}'$ is a coherent and minimal model of $(\Pi(D, IC))^{\mathcal{M}'}$. It just rests to prove that $D_{\mathcal{M}'} <_D D'$. First, we will prove $D_{\mathcal{M}'} \leq_D D'$. Let us suppose $P(\bar{a}) \in \Delta(D, D_{\mathcal{M}'})$. Then, either $P(\bar{a}) \in D$ and $P(\bar{a}) \notin D_{\mathcal{M}'}$ or $P(\bar{a}) \notin D$ and $P(\bar{a}) \in D_{\mathcal{M}'}$. In the first case, $P(\bar{a}, \mathbf{t_d})$, $P(\bar{a}, \mathbf{t}^\star)$ and $P(\bar{a}, \mathbf{f_a})$ are in $\mathcal{M}'$. These atoms are also in $\mathcal{M}$ and, given the only two ways in which $\mathcal{M}$ and $\mathcal{M}^\star_{IC}(D, D')$ can differ, they are also in $\mathcal{M}^\star_{IC}(D, D')$. Hence, $P(\bar{a}) \in \Delta(D, D')$. In the second case, $P(\bar{a}, \mathbf{t_a})$ and $P(\bar{a}, \mathbf{t}^\star)$ are in $\mathcal{M}'$. These atoms are also in $\mathcal{M}$ and, given the only two ways in which $\mathcal{M}$ and $\mathcal{M}^\star_{IC}(D, D')$ can differ, they are also in $\mathcal{M}^\star_{IC}(D, D')$. Hence, $P(\bar{a}) \in \Delta(D, D')$.

We will now prove $D_{\mathcal{M}'} <_D D'$. We know that, for some fact $P(\bar{a})$, $P\_(\bar{a}, \mathbf{t_a}) \in \mathcal{M}^\star_{IC}(D, D')$ and $P\_(\bar{a}, \mathbf{t_a}) \notin \mathcal{M}$, or $P\_(\bar{a}, \mathbf{f_a}) \in \mathcal{M}^\star_{IC}(D, D')$ and $P\_(\bar{a}, \mathbf{f_a}) \notin \mathcal{M}$. If $P(\bar{a}, \mathbf{f_a})$ is in $\mathcal{M}^\star_{IC}(D, D')$ and not in $\mathcal{M}$. Then, $P(\bar{a}) \in \Delta(D, D')$, but $P(\bar{a}) \notin \Delta(D, D_{\mathcal{M}'})$. Alternatively, if $P(\bar{a}, \mathbf{t_a})$ and $P(\bar{a}, \mathbf{t}^\star)$ are in $\mathcal{M}^\star_{IC}(D, D')$ and not in $\mathcal{M}$. Then, $P(\bar{a}) \in \Delta(D, D')$, but $P(\bar{a}) \notin \Delta(D, D_{\mathcal{M}'})$ and therefore $D_{\mathcal{M}'} <_D D'$. $\qquad\square$

Proposition 5.4 Given a database D and a set IC of UICs and RICs, if D' is a repair of D with respect to IC, then there is a stable model $\mathcal{M}$ of the program $(\Pi(D, IC))^{\mathcal{M}}$ such that $D_{\mathcal{M}} = D'$. Furthermore, the model $\mathcal{M}$ corresponds to $\mathcal{M}^\star_{IC}(D, D')$.

Proof: By Lemma 5.2, $\mathcal{M}^\star_{IC}(D, D')$ is a model of $\Pi(D, IC)^{\mathcal{M}^\star_{IC}(D,D')}$. We just have to show it is minimal. Let us suppose, by contradiction, that there exists a model $\mathcal{M}$ of $(\Pi(D, IC))^{\mathcal{M}^\star_{IC}(D,D')}$ such that $\mathcal{M} \subsetneq \mathcal{M}^\star_{IC}(D, D')$. Since $\mathcal{M} \subsetneq \mathcal{M}^\star_{IC}(D, D')$, the model $\mathcal{M}$ contains the atom $P(\bar{a}, \mathbf{t_d})$ iff $P(\bar{a}) \in D$. Then, we can assume, without loss of generality, that $\mathcal{M}$ is minimal (if it is not minimal, we can always generate from it a minimal model $\mathcal{M}'$, such that $\mathcal{M}' \subsetneq \mathcal{M}$, by deleting atoms that are not supported by program rules).

By Lemma 5.4, there exists model $\mathcal{M}'$ such that $D_{\mathcal{M}'} <_D D'$ and $\mathcal{M}'$ is a minimal model of $(\Pi(D, IC))^{\mathcal{M}'}$. By Lemma 5.3, $D_{\mathcal{M}'} \models_N IC$. This contradicts that D' is a repair. $\qquad\square$

Proposition 5.5 If $\mathcal{M}$ is a stable model of $\Pi(D, IC)$, then $D_{\mathcal{M}}$ is a repair of D with respect to IC.

Proof: From Lemma 5.3, we have $D_{\mathcal{M}} \models_N IC$. We only need to prove that it is $\leq_D$-*minimal*. Let us suppose there is a database instance D', such that D is a repair of D with respect to IC and $D' \leq_D D_{\mathcal{M}}$. From Proposition 5.4, $\mathcal{M}^{\star}_{IC}(D, D')$ is a stable model of $\Pi(D, IC)$ and $D_{\mathcal{M}^{\star}_{IC}(D,D')} = D'$.

For $D' \leq_D D_{\mathcal{M}}$ to hold, there should be an atom $P(\bar{a})$, with $\bar{a} \in \mathcal{U}$, in $\Delta(D, D_{\mathcal{M}})$ and not in $\Delta(D, D')$ or an atom $P(\bar{a}, \bar{b}) \in \Delta(D, D_{\mathcal{M}})$, with $\bar{a}', \bar{b} \in (\mathcal{U} \smallsetminus \{null\})$, and an atom $P(\bar{a}, null) \in \Delta(D, D')$.

1. $P(\bar{a}) \in \Delta(D, D_{\mathcal{M}})$ and $P(\bar{a}) \notin \Delta(D, D')$. Since $P(\bar{a}) \in \Delta(D, D_{\mathcal{M}})$ we have that $P(a, \mathbf{t_a})$ or $P(a, \mathbf{f_a})$ belongs to $\mathcal{M}$. By Lemma 5.1, there are two options:

   - $P(\bar{a}, \mathbf{t_d})$, $P(\bar{a}, \mathbf{t^{\star}})$ and $P(\bar{a}, \mathbf{f_a})$ belong to $\mathcal{M}$, and no other $P(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$. $P(\bar{a}, \mathbf{t_d})$, $P(\bar{a}, \mathbf{t^{\star}})$ and $P(\bar{a}, \mathbf{t^{\star\star}})$ belong to $\mathcal{M}^{\star}$, and no other $P(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}^{\star}$.

   - $P(\bar{a}, \mathbf{t_a})$, $P(\bar{a}, \mathbf{t^{\star}})$ and $P(\bar{a}, \mathbf{t^{\star\star}})$ belong to $\mathcal{M}$, and no other $P(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}$. No $P(\bar{a}, v)$, for v an annotation value, belongs to $\mathcal{M}^{\star}$.

   If an atom belongs to a model $\mathcal{M}_1$, e.g. $P(\bar{a}, \mathbf{f_a})$, and there is another model $\mathcal{M}_2$ in which it is not present, then there must be in $\mathcal{M}_2$ an atom annotated with $\mathbf{t_a}$ or $\mathbf{f_a}$ in order to satisfy the rule that was satisfied in $\mathcal{M}_1$ by $P(\bar{a}, \mathbf{f_a})$. This implies that $\mathcal{M}^{\star}$ has an atom annotated with $\mathbf{t_a}$ or $\mathbf{f_a}$ that does not belong to

$\mathcal{M}$. This implies that there is an atom that belongs to $\Delta(D, D')$ and that does not belong to $\Delta(D, D_\mathcal{M})$. We have reached a contradiction, because $\Delta(D, D')$ is a proper subset of $\Delta(D, D_\mathcal{M})$.

2. $P(\bar{a}, \bar{b}) \in \Delta(D, D_\mathcal{M})$ and $P(\bar{a}, null) \in \Delta(D, D')$. If $\mathcal{M} \not\models_N P(\bar{a}', \bar{b})$, then $\mathcal{M} \models_N P_{\_}(\bar{a}', \bar{b}, \mathbf{t_a})$, $\mathcal{M} \not\models_N P(\bar{a}', \overline{null})$ and $\mathcal{M} \not\models_N P_{\_}(\bar{a}', \overline{null}, \mathbf{t_a})$. Since $P(\bar{a}, null) \in \Delta(D, D')$ and $\mathcal{M} \not\models_N P(\bar{a}', \overline{null})$, $\mathcal{M}^\star \models_N P_{\_}(\bar{a}', \overline{null}, \mathbf{t_a})$. Since $\mathcal{M}^\star \models_N P_{\_}(\bar{a}', \overline{null}, \mathbf{t_a})$, there must be a rule representing a RIC in $\Pi(D, IC)$ such that $P_{\_}(\bar{a}', \overline{null}, \mathbf{t_a})$ is the only atom true in the head. For that rule to also be satisfied by $\mathcal{M}$, there must be another atom in the head of that rule which is true in $\mathcal{M}$ and not in $\mathcal{M}^\star$. This means there is a $P(\bar{b}) \in \Delta(D, D_\mathcal{M})$ and $P(\bar{b}) \notin \Delta(D, D')$. This brings us back to alternative 1. which was proven to lead to a contradiction.

Therefore, it is not possible to have $D' <_D D_\mathcal{M}$; and $D_\mathcal{M}$ is a repair of D. $\quad\square$

Proof of Theorem 5.4: Directly from Propositions 5.4 and 5.5. $\quad\square$

We have proven that the program $\Pi(D, IC)$ can be used to compute the repairs of a database with respect to a RIC-acyclic set of UICs and RICs. If the set of ICs is not RIC-acyclic, then we will have some models whose associated databases will be consistent with respect to IC but will not be repairs (they will not be $<_D$-$minimal$).

Example 5.12 Consider the database $\{P(a, b, c), Q(b, c), Q(a, a)\}$ and the following cyclic set of ICs: $\{\forall xyz(P(x, y, z) \to \exists v Q(y, v)), \forall xy(Q(x, y) \to \exists u P(u, x, y))\}$. The expected repairs are $\{P(a, b, c), Q(b, c)\}$ and $\{P(a, b, c), Q(b, c), Q(a, a), P(null, a, a)\}$. The program $\Pi(D, IC)$ corresponds to:

$dom(a).\quad dom(b).\quad dom(c).\quad P_{\_}(a, b, c, \mathbf{t_d}).\quad Q_{\_}(b, c, \mathbf{t_d}).\quad Q_{\_}(a, a, \mathbf{t_d}).$

$P_{\neg}(x, y, z, \mathbf{f_a}) \vee Q_{\neg}(y, null, \mathbf{t_a}) \leftarrow P_{\neg}(x, y, z, \mathbf{t^\star}), \; not \; aux_1(y), \; dom(y).$

$aux_1(x) \leftarrow Q_{\neg}(x, y, \mathbf{t^\star}), \; not \; Q_{\neg}(x, y, \mathbf{f_a}), \; dom(x), \; dom(y).$

$aux_1(x) \leftarrow Q_{\neg}(x, null, \mathbf{t_d}), \; not \; Q_{\neg}(x, null, \mathbf{f_a}), \; dom(x).$

$Q_{\neg}(x, y, \mathbf{f_a}) \vee P_{\neg}(null, x, y, \mathbf{t_a}) \leftarrow Q_{\neg}(x, y, \mathbf{t^\star}), \; not \; aux_2(x, y), \; dom(x), \; dom(y).$

$aux_2(y, z) \leftarrow P_{\neg}(x, y, z, \mathbf{t^\star}), \; not \; P_{\neg}(x, y, z, \mathbf{f_a}), \; dom(x), \; dom(y), \; dom(z).$

$aux_2(y, z) \leftarrow P_{\neg}(null, y, z, \mathbf{t_d}), \; not \; P_{\neg}(null, y, z, \mathbf{f_a}), \; dom(y), \; dom(z).$

$$
\left.
\begin{array}{l}
P_{\neg}(x, y, z, \mathbf{t^\star}) \leftarrow P_{\neg}(x, y, z, \mathbf{t_a}). \\[4pt]
P_{\neg}(x, y, z, \mathbf{t^\star}) \leftarrow P_{\neg}(x, y, z, \mathbf{t_d}). \\[4pt]
P_{\neg}(x, y, z, \mathbf{f^\star}) \leftarrow P_{\neg}(x, y, z, \mathbf{f_a}). \\[4pt]
P_{\neg}(x, y, z, \mathbf{f^\star}) \leftarrow dom(x), dom(y), dom(z), \; not \; P_{\neg}(x, y, z, \mathbf{t_d}). \\[4pt]
P_{\neg}(x, y, z, \mathbf{t^{\star\star}}) \leftarrow P_{\neg}(x, y, z, \mathbf{t_a}). \\[4pt]
P_{\neg}(x, y, z, \mathbf{t^{\star\star}}) \leftarrow P_{\neg}(x, y, z, \mathbf{t_d}), \; not \; P_{\neg}(x, y, z, \mathbf{f_a}). \\[4pt]
P_{\neg}(x, y, z, \mathbf{f^{\star\star}}) \leftarrow P_{\neg}(x, y, z, \mathbf{f_a}). \\[4pt]
P_{\neg}(x, y, z, \mathbf{f^{\star\star}}) \leftarrow dom(x), \; dom(y), \; dom(z), \; not \; P_{\neg}(x, y, z, \mathbf{t_d}), \\[4pt]
\qquad\qquad\qquad\qquad not \; P_{\neg}(x, y, z, \mathbf{t_a}).
\end{array}
\right\} \quad \text{(Similarly for } Q\text{)}
$$

$\leftarrow P_{\neg}(x, y, z, \mathbf{t_a}), P_{\neg}(x, y, z, \mathbf{f_a}).$

$\leftarrow Q_{\neg}(x, y, \mathbf{t_a}), Q_{\neg}(x, y, \mathbf{f_a}).$

$\Pi(D, IC)$ will compute models that satisfy the ICs but are not necessarily minimal. In this case, we get the following databases associated to its models:

$D_{\mathcal{M}_1} = \{P(a, b, c), Q(b, c), Q(a, a), P(null, a, a)\}$

$D_{\mathcal{M}_2} = \{P(a, b, c), Q(b, c)\}$

$D_{\mathcal{M}_3} = \{P(null, a, a), Q(a, a)\}$

$D_{\mathcal{M}_4} = \emptyset$

The first two correspond to the repairs of D with respect to IC. The last two are consistent, but are not repairs since $D_{\mathcal{M}_1} \leq_D D_{\mathcal{M}_3}$ and $D_{\mathcal{M}_2} \leq_D D_{\mathcal{M}_4}$. The databases

associated to the models of $\Pi(D, IC)$ are superset of the repairs. $\qquad\square$

It can be shown that, for a database D and a RIC-cyclic set of constraints IC, the program captures a superset of the repairs of D with respect to IC.

The repair program $\Pi(D, IC)$ can be optimized as shown in Appendix A. There, materialization of negative information is avoided and the predicate *dom* is deleted. The next definition corresponds to the optimized version of the program.

Definition 5.9 Given a database instance D, a set IC of UICs, RICs and NNCs, the repair program $\Pi^\star(D,\ IC)$ contains the following rules:

1. Facts: $P(\bar{a})$ for each atom $P(\bar{a}) \in D$.

2. For every UIC ψ of the form (2.2), the rules:

   $$\bigvee_{i=1}^{n} P_{i\text{-}}(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} Q_{j\text{-}}(\bar{y}_j, \mathbf{t_a}) \ \leftarrow \ \bigwedge_{i=1}^{n} P_{i\text{-}}(\bar{x}_i, \mathbf{t^\star}), \bigwedge_{Q_{j\text{-}} \in Q'} Q_{j\text{-}}(\bar{y}_j, \mathbf{f_a}),$$
   $$\bigwedge_{Q_k \in Q''} \ not \ Q_k(\bar{y}_k), \bigwedge_{x_l \in \mathcal{A}(\psi)} x_l \neq null, \bar{\varphi}.$$

   for every set Q' and Q'' of atoms appearing in formula (2.2) such that $Q' \cup Q'' = \bigcup_{j=1}^{m} Q_j(\bar{y}_j)$ and $Q' \cap Q'' = \emptyset$.[6] Here, $\mathcal{A}(\psi)$ is the set of relevant attributes for ψ (see Definition 4.8), $\bar{x} = \bigcup_{i=1}^{n} x_i$ and $\bar{\varphi}$ is a conjunction of built-ins that is equivalent to the negation of φ.

3. For every RIC ψ of the form (2.3), the rules:

   $P_{\text{-}}(\bar{x}, \mathbf{f_a}) \vee Q_{\text{-}}(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow P_{\text{-}}(\bar{x}, \mathbf{t^\star}), \ not \ \ aux_\psi(\bar{x}'), \bar{x}' \neq null.$

   $aux_\psi(\bar{x}') \leftarrow Q(\bar{x}', \overline{null}), \ \ not \ Q_{\text{-}}(\bar{x}', \overline{null}, \mathbf{f_a}), \bar{x}' \neq null.$

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

   $aux_\psi(\bar{x}') \leftarrow Q_{\text{-}}(\bar{x}', \bar{y}, \mathbf{t^\star}), \ \ not \ Q_{\text{-}}(\bar{x}', \bar{y}, \mathbf{f_a}), \bar{x}' \neq null, y_i \neq null.$

---

[6]We are assuming in this definition that the rules are a direct translation of the original ICs introduced in Section 2; in particular, the same variables are used and the standardization conditions about their occurrences are respected in the program.

4. For each predicate $P \in R$, the annotation rules:

$$P_{-}(\bar{x}, \mathbf{t}^{\star}) \leftarrow P(\bar{x}).$$
$$P_{-}(\bar{x}, \mathbf{t}^{\star}) \leftarrow P_{-}(\bar{x}, \mathbf{t_a}).$$

5. For every predicate $P \in \mathcal{R}$, the interpretation rule:

$$P_{-}(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P_{-}(\bar{x}, \mathbf{t}^{\star}), \;\; not \; P_{-}(\bar{x}, \mathbf{f_a}).$$

6. For every predicate $P \in \mathcal{R}$, the program denial constraint:

$$\leftarrow \; P_{-}(\bar{x}, \mathbf{t_a}), \, P_{-}(\bar{x}, \mathbf{f_a}). \hspace{3cm} \Box$$

The set of predicates Q' and Q'' are used to check that in all the possible combinations, the consequent of a UIC is not being satisfied. This set of rules correspond to applying the transformation rules used to replace the predicates containing $\mathbf{f}^{\star}$ and $\mathbf{f}^{\star\star}$. The predicate $dom(x)$ was deleted and its occurrence replaced in all rules by $x \neq null$.

In the rest of the thesis, we will continue using the optimized version of the program. Therefore, we will denote $\Pi^{\star}(D, IC)$ simply by $\Pi(D, IC)$.

5.2.1 Consistent Answers

The semantics of consistent query answering (Definition 5.5) depends on which query answering semantics we choose for databases with null values. We will give techniques to obtain CQAs using the repair program for the null query answering semantics introduced in Definition 4.15 in Chapter 4 and for the SQL query answering semantics [International Organization for Standardization, 2003].

Null Consistent Query Answers

Definition 5.5 provides the general definition of consistent query answers on the basis of a query answering semantics $\models^{\mathcal{Q}}$. In this section, we want to use the null query answering semantics introduced in Definition 4.15 of Section 4.2.

Definition 5.10 Given a database D, a set of ICs IC, and a query $Q(\bar{x})$, a ground tuple $\bar{t}$ is a *null consistent answer* to Q with respect to IC in D iff for every $D' \in Rep(D, IC)$, $D' \models^q_N Q[\bar{t}]$. If Q is a sentence (boolean query), then *yes* is a null consistent answer iff $D' \models^q_N Q$ for every $D' \in Rep(D, IC)$. Otherwise, the consistent answer is *no*. $\qquad\qquad\square$

In order to obtain null consistent answers to a FO query Q from a database D, we can use the repair program $\Pi(D, IC)$ in the following way:

1. Query Q is transformed into a query written as a logic program $\Pi(Q)$. This transformation is done using a standard process [Lloyd, 1987; Abiteboul *et al.*, 1995] and $\Pi(Q)$ so obtained is a stratified Datalog programs [Abiteboul *et al.*, 1995]. The query program has a query predicate *Ans* that collects the answers to Q.

2. Each positive occurrence, say $P(\bar{t})$, of a database predicate in $\Pi(Q)$ is replaced by $P_-(\bar{t}, \mathbf{t}^{\star\star})$.

3. Replace every rule $(H \leftarrow B) \in \Pi(Q)$ by: $(H \leftarrow (B \wedge \bigwedge_{v \in \mathcal{V}(Q)} v \neq null))$. This ensures that relevant variables are evaluated only over values in $(\mathcal{U} \smallsetminus \{null\})$.

4. Program $\Pi(Q)$ is appended to the repair program $\Pi(D, IC)$.

5. The null consistent answers to Q are the ground *Ans* atoms in the intersection of all stable models of $\Pi(Q) \cup \Pi(D, IC)$. If Q is a boolean query, *yes* is a null consistent answer if *Ans* (a propositional atom) is in the intersection of all the models, and *no* if it is not.

Example 5.13 Consider the ICs $IC = \{\forall xy(P(x, y) \rightarrow R(x)), \forall x(T(x) \rightarrow \exists y\, P(x, y))\}$ and an inconsistent database $D = \{P(a, b), P(null, a), T(c), R(null)\}$. Query

$\mathcal{Q}_1$: $\exists x P(x, y)$ has no relevant variables. Therefore the query is rewritten as the logic program $\Pi(Q_1)$: $Ans(y) \leftarrow P_{\text{-}}(x, y, \mathbf{t}^{\star\star})$. The program $\Pi(D, IC)$ has four stable models:[7]

$$
\begin{aligned}
\mathcal{M}_1 = \ & \{P_{\text{-}}(a, b, \mathbf{t}^\star),\ P_{\text{-}}(null, a, \mathbf{t}^\star),\ T_{\text{-}}(c, \mathbf{t}^\star),\ R_{\text{-}}(null, \mathbf{t}^\star),\ \underline{P_{\text{-}}(a, b, \mathbf{t}^{\star\star})},\ \underline{P_{\text{-}}(null,} \\
& \underline{a, \mathbf{t}^{\star\star})},\ T_{\text{-}}(c, \mathbf{f_a}),\ R_{\text{-}}(a, \mathbf{t_a}),\ \underline{R_{\text{-}}(null, \mathbf{t}^{\star\star})},\ R_{\text{-}}(a, \mathbf{t}^\star),\ \underline{R_{\text{-}}(a, \mathbf{t}^{\star\star})},\ aux(a)\}, \\[4pt]
\mathcal{M}_2 = \ & \{P_{\text{-}}(a, b, \mathbf{t}^\star),\ P_{\text{-}}(null, a, \mathbf{t}^\star),\ T_{\text{-}}(c, \mathbf{t}^\star),\ R_{\text{-}}(null, \mathbf{t}^\star),\ \underline{P_{\text{-}}(a, b, \mathbf{t}^{\star\star})},\ \underline{P_{\text{-}}(null,} \\
& \underline{a, \mathbf{t}^{\star\star})},\ \underline{T_{\text{-}}(c, \mathbf{t}^{\star\star})},\ R_{\text{-}}(a, \mathbf{t_a}),\ P_{\text{-}}(c, null, \mathbf{t_a}),\ P_{\text{-}}(c, null, \mathbf{t}^\star),\ R_{\text{-}}(a, \mathbf{t}^\star), \\
& \underline{P_{\text{-}}(c, null, \mathbf{t}^{\star\star})},\ \underline{R_{\text{-}}(a, \mathbf{t}^{\star\star})},\ \underline{R_{\text{-}}(null, \mathbf{t}^\star)},\ R_{\text{-}}(c, \mathbf{t_a}),\ R_{\text{-}}(c, \mathbf{t}^\star),\ \underline{R_{\text{-}}(c, \mathbf{t}^{\star\star})}, \\
& aux(a)\}, \\[4pt]
\mathcal{M}_3 = \ & \{P_{\text{-}}(a, b, \mathbf{t}^\star),\ P_{\text{-}}(null, a, \mathbf{t}^\star),\ T_{\text{-}}(c, \mathbf{t}^\star),\ R_{\text{-}}(null, \mathbf{t}^\star),\ P_{\text{-}}(a, b, \mathbf{f_a}),\ \underline{P_{\text{-}}(null,} \\
& \underline{a, \mathbf{t}^{\star\star})},\ \underline{T_{\text{-}}(c, \mathbf{t}^{\star\star})},\ P_{\text{-}}(c, null, \mathbf{t_a}),\ P_{\text{-}}(c, null, \mathbf{t}^\star),\ \underline{P_{\text{-}}(c, null, \mathbf{t}^{\star\star})},\ R_{\text{-}}(c, \mathbf{t_a}), \\
& R_{\text{-}}(c, \mathbf{t}^\star),\ \underline{R_{\text{-}}(null, \mathbf{t}^{\star\star})},\ \underline{R_{\text{-}}(c, \mathbf{t}^{\star\star})}\}, \\[4pt]
\mathcal{M}_4 = \ & \{P_{\text{-}}(a, b, \mathbf{t}^\star),\ P_{\text{-}}(null, a, \mathbf{t}^\star),\ T_{\text{-}}(c, \mathbf{t}^\star),\ R_{\text{-}}(null, \mathbf{t}^\star),\ P_{\text{-}}(a, b, \mathbf{f_a}),\ \underline{P_{\text{-}}(null,} \\
& \underline{a, \mathbf{t}^{\star\star})},\ \underline{R_{\text{-}}(null, \mathbf{t}^{\star\star})},\ T_{\text{-}}(c, \mathbf{f_a})\}.
\end{aligned}
$$

Then, the repairs are: $D_{\mathcal{M}_1} = \{P(a, b), P(null, a), R(a), R(null)\}$, $D_{\mathcal{M}_2} = \{P(a, b), P(null, a), T(c), P(c, null), R(a), R(c), R(null)\}$, $D_{\mathcal{M}_3} = \{P(null, a), T(c), P(c, null), R(c), R(null)\}$, and $D_{\mathcal{M}_4} = \{P(null, a), R(null)\}$. The stable models of program $(\Pi(D, IC) \cup \Pi(Q_1))$ are the stable models of $\Pi(D, IC)$ expanded by the answers to the query:

$$
\begin{aligned}
\mathcal{M}_1 = \ & \mathcal{M}_1 \cup \{Ans(b), Ans(a)\}, \\
\mathcal{M}_2 = \ & \mathcal{M}_2 \cup \{Ans(b), Ans(a)\}, \\
\mathcal{M}_3 = \ & \mathcal{M}_3 \cup \{Ans(a), Ans(null)\}, \\
\mathcal{M}_4 = \ & \mathcal{M}_4 \cup \{Ans(a)\}.
\end{aligned}
$$

Since the null consistent answers are those we get from all the possible repairs, the only answer to this query is $\{(a)\}$.

---

[7]Facts and atoms annotated with $\mathbf{f}^\star$ and $\mathbf{f}^{\star\star}$ are omitted from the models.

On the other hand, for query $Q_2 : \exists xy(P(x,y) \wedge R(x))$, with $\mathcal{V}(Q_2) = \{x\}$, the query program $\Pi(Q_2)$ is: $Ans \leftarrow P_{\text{-}}(x,y,\mathbf{t}^{\star\star}) \wedge R_{\text{-}}(x,\mathbf{t}^{\star\star}) \wedge x \neq null$. The stable models of program $(\Pi(D,IC) \cup \Pi(\mathcal{Q}_1))$ are the stable models of $\Pi(D,IC)$ expanded by the answers to the query:

$$\mathcal{M}_1 = \mathcal{M}_1 \cup \{Ans\},$$
$$\mathcal{M}_2 = \mathcal{M}_2 \cup \{Ans\},$$
$$\mathcal{M}_3 = \mathcal{M}_3 \cup \{Ans\},$$
$$\mathcal{M}_4 = \mathcal{M}_4 \cup \{\}.$$

The consistent answer to this boolean query is *no*. □

If a given database D is consistent with respect to a set of ICs, then there is only one model and one associated repair, that coincides with D. We have successfully experimented with CQA based on this specification of database repairs using the DLV system [Leone *et al.*, 2006].

SQL Consistent Query Answering

Now we want to modify Definition 5.5 of consistent query answers so that the query answering semantics, $\models^{\mathcal{Q}}$, is the SQL query answering semantics [International Organization for Standardization, 2003].

Definition 5.11 Given a database D, a set of ICs IC, and a query $Q(\bar{x})$, a ground tuple $\bar{t}$ is a *SQL-consistent answer* to Q with respect to IC in D iff for every $D' \in Rep(D,IC)$, $D' \models_N^{SQL} Q[\bar{t}]$. If Q is a sentence (boolean query), then *yes* is a SQL consistent answer iff $D' \models_N^{SQL} Q$ for every $D' \in Rep(D,IC)$. Otherwise, the consistent answer is *no*. □

In order to obtain null consistent answers to extended conjunctive query (ECQ) Q from a database D we can use the results of Proposition 4.3. The SQL-consistent

answers to query Q can be computed as the null consistent answers to query $Q^{\mathfrak{N}}$ (see Definition 4.19). This is of particular relevance since this is the semantics used in commercial databases. Further research is needed in finding more relations between the null query answering semantics and the SQL query answering semantics, so that we can get answers under the latter from logic programs.

Example 5.14 (example 5.13 continued) For query $Q_3 : \exists x (P(x, y) \wedge \neg R(x))$, the rewritten query $Q_3^{\mathfrak{N}}$ is: $(\exists x (P(x, y) \wedge \neg R(x)) \vee P(null, y))$. The SQL-consistent answers of Q_3 are the null consistent answers of $Q_3^{\mathfrak{N}}$. They can be obtained via the query program $\Pi(Q_3^{\mathfrak{N}})$:

$Ans(y) \leftarrow P\_(x, y, \mathbf{t}^{\star\star}), \ not \ R\_(x, \mathbf{t}^{\star\star}), x \neq null.$

$Ans(y) \leftarrow P\_(null, y, \mathbf{t}^{\star\star}).$

The stable models of program $(\Pi(D, IC) \cup \Pi(Q_3^{\mathfrak{N}}))$ are the stable models of $\Pi(D, IC)$ expanded by the null consistent answers to the query $Q_3^{\mathfrak{N}}$:

$$\mathcal{M}_1 = \mathcal{M}_1 \cup \{Ans(a)\},$$
$$\mathcal{M}_2 = \mathcal{M}_2 \cup \{Ans(a)\},$$
$$\mathcal{M}_3 = \mathcal{M}_3 \cup \{Ans(a)\},$$
$$\mathcal{M}_4 = \mathcal{M}_4 \cup \{Ans(a)\}.$$

The only null consistent answer to query $Q_3^{\mathfrak{N}}$ is $\{(a)\}$ and therefore the SQL consistent answers to Q_3 is $\{(a)\}$. $\qquad\square$

5.2.2 Not Null Constraints

So far we have considered CQA for ICs of the form (2.1) and have not considered the effect of having constraints with the special predicate *IsNull*. In particular, we have not considered not null constraints (NNC), as introduced in Definition 4.5.

Example 5.15 Consider a schema with relations $R(X, Y)$, with primary key (PK) $R[1]$, and $S(U, V)$, with $S[2]$ a foreign key (FK) to table R, i.e., $S[2] \subseteq R[1]$. Then, as discussed in Section 4.1.2, the PK can be written as $\forall xyz(R(x, y) \wedge R(x, z) \rightarrow y = z)$, $\forall xyz(R(x, y) \wedge R(x, z) \wedge IsNull(x) \rightarrow IsNull(y))$ and the NNC $\forall xy(R(x, y) \wedge IsNull(x) \rightarrow \textbf{false})$; and the FK as $\forall uv(S(u, v) \rightarrow \exists y\ R(v, y))$. Since there is no constraint with an existential quantifier over $R[1]$, the NNC will not be violated while solving inconsistencies with respect to the FK. We would have, then, a *non-conflicting interaction* of RICs and NNCs. Database $D = \{R(a, b), R(a, c), R(null, a), S(e, f), S(null, a)\}$ is inconsistent, since the PK is violated by $\{R(a, b), R(a, c)\}$, and $\{R(null, a)\}$; and the FK is violated by $S(e, f)$. The repairs are:

$D_1 = \{R(a, b),\ S(e, f),\ S(null, a),\ R(f, null)\}$,

$D_2 = \{R(a, c),\ S(e, f),\ S(null, a),\ R(f,\ null)\}$,

$D_3 = \{R(a, b),\ S(null, a)\}$, and

$D_4 = \{R(a, c),\ S(null, a)\}$. □

The following example shows what can happen if we have a *conflicting interaction* of a RIC containing an existential quantifier over a certain variable with an additional NNC that prevents that variable from tacking the null value.

Example 5.16 Consider the database $D = \{P(a), P(b), Q(b, c)\}$, the RIC $\forall x(P(x) \rightarrow \exists y\ Q(x, y))$, and the NNC $\forall xy(Q(x, y) \wedge IsNull(y) \rightarrow \textbf{false})$ over an existentially quantified attribute in the RIC. We cannot repair as expected using *null*. Actually, the repairs, as introduced in Definitions 5.2 and 5.3, are $\{P(b), Q(b, c)\}$, corresponding to a tuple deletion, and also those of the form $\{P(a), P(b), Q(b, c), Q(a, \mu)\}$, for every $\mu \in (\mathcal{U} \smallsetminus \{null\})$, that are obtained by tuple insertions. We thus recover the repair semantics of [Arenas *et al.*, 1999] given in Definition 5.1. □

With a conflicting interaction of RICs and NNCs we could have infinitely many repairs (see remark after Example 5.6).

The repair semantics of Definitions 5.2 and 5.3 could be modified so that the repairs are obtained only through tuple deletions when *null* cannot be used to repair due to the presence of conflicting NNCs. This could be done as follows. If $Rep(D, IC)$ is the class of repairs according to Definition 5.3, the alternative class of repairs, $Rep_d(D, IC)$, that prefer tuple deletions over insertions with arbitrary non-null elements of the domain when there are of conflicting NNCs, can be defined by $Rep_d(D, IC) := \{D' \mid D' \in Rep(D, IC) \text{ and there is no } D'' \in Rep(D, IC') \text{ with } D'' <_D D'\}$, where IC' is IC without the (conflicting) NNCs.

In the following, we will continue using the repairs introduced in Definitions 5.2 and 5.3; and in order to avoid repairs like those in Example 5.16, we will make the following assumption:

Assumption: Our sets of ICs of the form (2.1) and NNCs, are *non-conflicting*, in the sense that there is no NNC on an attribute that is existentially quantified in an IC of the form (2.1).

In this way, we will always be able to repair RICs by tuple deletions or tuple insertions with *null*. Notice that every set of ICs consisting of primary key constraints, foreign key constraints, and check constraints satisfies this condition. Also note that if there are non-conflicting NNCs, the given semantics and the one based on Rep_d-repairs coincide.

It can be easily proven that Proposition 5.1 and Theorems 5.1, 5.2 and 5.3 still hold if we add non-conflicting NNC. That is, the problem of determining if a database is a repair of a database with respect to a set of ICs of the form (2.1) and NNC of the form (4.2) is *coNP*-complete. Also, consistent query answering for first-order queries

and sets of ICs (including only non-conflicting NNCs) is Π_2^P-complete.

Now, we need to extend the repair program so that we can include NNCs. The following definition of repair program also includes the optimizations introduced in the previous section.

Definition 5.12 Given a database instance D, a set IC of UICs, RICs and NNCs, the repair program $\Pi(D, IC)$ contains the following rules:

1. Same rules as in Definition 5.9.

2. For every NNC of the form (4.2), the rule:

   $P\_(\bar{x}, \mathbf{f_a}) \leftarrow P\_(\bar{x}, \mathbf{t^\star}),\ x_i = null.$ □

Rule 2. captures, in the antecedent of the rule, the violation of ICs of the form (4.2) and, with the consequent of the rule, the intended way of restoring consistency.

Example 5.17 (example 5.15 continued) The repair program $\Pi(D, IC)$ is the following:

1. $R(a, b). \quad R(a, c). \quad R(null, a). \quad S(e, f). \quad S(null, a).$

2. $R\_(x, y, \mathbf{f_a}) \vee R\_(x, z, \mathbf{f_a}) \ \leftarrow\ R\_(x, y, \mathbf{t^\star}), R\_(x, z, \mathbf{t^\star}), y \neq z, x \neq null.$

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

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

4. $R\_(x, y, \mathbf{f_a}) \leftarrow R\_(x, y, \mathbf{t^\star}), x = null.$

5. $R\_(x, y, \mathbf{t^\star}) \leftarrow R\_(x, y, \mathbf{t_a}).$

   $R\_(x, y, \mathbf{t^\star}) \leftarrow R(x, y).$

   $R\_(x, y, \mathbf{t^{\star\star}}) \leftarrow R\_(x, y, \mathbf{t_a}).$ (Similarly for S)

   $R\_(x, y, \mathbf{t^{\star\star}}) \leftarrow R(x, y), \ not\ R\_(x, y, \mathbf{f_a}).$

   $\leftarrow R\_(x, y, \mathbf{t_a}), R\_(x, y, \mathbf{f_a}).$

Rules 2., 3. and 4. depend on the ICs: rule 2. for the UIC, 3. for the RIC and 4. for the NNC. They say how to repair the inconsistencies. In rule 2., $Q' = Q'' = \emptyset$, because there is no database predicate in the consequent of the UIC, therefore there is only one rule.

The program has four stable models (the facts of the program are omitted for simplicity):

$$\mathcal{M}_1 = \{R_\text{-}(a, b, \mathbf{t}^\star),\ R_\text{-}(a, c, \mathbf{t}^\star),\ R_\text{-}(null, a, \mathbf{t}^\star),\ S_\text{-}(e, f, \mathbf{t}^\star),\ S_\text{-}(null, a, \mathbf{t}^\star),\ aux(a),$$
$$R_\text{-}(null, a, \mathbf{f_a}),\ \underline{S_\text{-}(e, f, \mathbf{t}^{\star\star})},\ \underline{S_\text{-}(null, a, \mathbf{t}^{\star\star})},\ R_\text{-}(f, null, \mathbf{t_a}),\ \underline{R_\text{-}(a, b, \mathbf{t}^{\star\star})},$$
$$R_\text{-}(a, c, \mathbf{f_a}),\ R_\text{-}(f, null, \mathbf{t}^\star),\ \underline{R_\text{-}(f, null, \mathbf{t}^{\star\star})}\ \},$$

$$\mathcal{M}_2 = \{R_\text{-}(a, b, \mathbf{t}^\star),\ R_\text{-}(a, c, \mathbf{t}^\star),\ R_\text{-}(null, a, \mathbf{t}^\star),\ S_\text{-}(e, f, \mathbf{t}^\star),\ S_\text{-}(null, a, \mathbf{t}^\star),\ aux(a),$$
$$R_\text{-}(null, a, \mathbf{f_a}),\ \underline{S_\text{-}(e, f, \mathbf{t}^{\star\star})},\ \underline{S_\text{-}(null, a, \mathbf{t}^{\star\star})},\ R_\text{-}(f, null, \mathbf{t_a}),\ R_\text{-}(a, b, \mathbf{f_a}),$$
$$\underline{R_\text{-}(a, c, \mathbf{t}^{\star\star})},\ R_\text{-}(f, null, \mathbf{t}^\star),\ \underline{R_\text{-}(f, null, \mathbf{t}^{\star\star})}\ \},$$

$$\mathcal{M}_3 = \{R_\text{-}(a, b, \mathbf{t}^\star),\ R_\text{-}(a, c, \mathbf{t}^\star),\ R_\text{-}(null, a, \mathbf{t}^\star),\ S_\text{-}(e, f, \mathbf{t}^\star),\ S_\text{-}(null, a, \mathbf{t}^\star),\ aux(a),$$
$$R_\text{-}(null, a, \mathbf{f_a}),\ S_\text{-}(e, f, \mathbf{f_a}),\ \underline{S_\text{-}(null, a, \mathbf{t}^{\star\star})},\ \underline{R_\text{-}(a, b, \mathbf{t}^{\star\star})},\ R_\text{-}(a, c, \mathbf{f_a})\},$$

$$\mathcal{M}_4 = \{R_\text{-}(a, b, \mathbf{t}^\star),\ R_\text{-}(a, c, \mathbf{t}^\star),\ R_\text{-}(null, a, \mathbf{t}^\star),\ S_\text{-}(e, f, \mathbf{t}^\star),\ S_\text{-}(null, a, \mathbf{t}^\star),\ aux(a),$$
$$R_\text{-}(null, a, \mathbf{f_a}),\ S_\text{-}(e, f, \mathbf{f_a}),\ \underline{S_\text{-}(null, a, \mathbf{t}^{\star\star})},\ R_\text{-}(a, b, \mathbf{f_a}),\ \underline{R_\text{-}(a, c, \mathbf{t}^{\star\star})}\}.$$

The databases associated to the models select the underlined atoms: $D_{\mathcal{M}_1} = \{S(e, f), S(null, a),\ R(a, b),\ R(f, null)\}$, $D_{\mathcal{M}_2} = \{S(e, f),\ S(null, a),\ R(a, c),\ R(f,\ null)\}$, $D_{\mathcal{M}_3} = \{S(null, a), R(a, b)\}$, and $D_{\mathcal{M}_4} = \{S(null, a), R(a, c)\}$. As expected, these are the repairs obtained in Example 5.15. $\qquad\square$

This repair program specifies the Rep_d-repairs, i.e., if there is a set of ICs with conflicting NNC, then the program will solve the inconsistencies by deletion instead of by adding tuples with all possible values in the existentially quantified variables.

Theorem 5.5 Let IC be a RIC-acyclic set of UICs, RICs and a set of non-conflicting NNCs. If $\mathcal{M}$ is a stable model of $\Pi(D, IC)$, then $D_\mathcal{M}$ is a repair of D with respect

to IC. Furthermore, the repairs obtained in this way are all the repairs of D. $\square$

Proof: In order to prove this theorem, we need to use and modify some lemmas and propositions of Section 5.2. Lemma 5.1 can be extended to consider non-conflicting NNC without modifying the proof. On the other hand, the proof of Lemma 5.2 needs to be modified to consider non-conflicting NNC by adding before the last sentence: "For every NNC in IC there is a rule of the form 2. If the body of the rule is true, e.g. $P(\bar{a}, null, \mathbf{t}^{\star}) \in \mathcal{M}_{IC}^{\star}(D, D')$, the constraint is not satisfied at some point in the repair process. Since it is satisfied in D', $P(\bar{a}, null) \notin D'$."

Since Lemma 5.1 holds when non-conflicting NNC are added to the set of IC, we can prove Lemma 5.3 by adding the following case to the proof:

- Formula ψ is a NNC. Since $\mathcal{M}$ is a model of $(\Pi(D, IC))^{\mathcal{M}}$, $\mathcal{M}$ satisfies rules 2 of $\Pi(D, IC)$ (see Definition 5.12). Then, at least one of the following cases holds:

    - $\mathcal{M} \models_{N} P(\bar{a}, \mathbf{f_a})$. Then, $\mathcal{M} \not\models_{N} P(\bar{a}, \mathbf{t}^{\star\star})$ and $P(\bar{a}) \notin D_{\mathcal{M}}$ (by Lemma 5.1). Hence, $D_{\mathcal{M}} \models_{N} \neg P(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_{\mathcal{M}}$ satisfies the constraint.

    - $\mathcal{M} \not\models_{N} P(\bar{a}, \mathbf{t}^{\star})$. Given the model is minimal, just the last item in Lemma 5.1 holds. This means $\mathcal{M} \not\models_{N} P(\bar{a}, \mathbf{t}^{\star\star})$, $P(\bar{a}) \notin D_{\mathcal{M}}$ and $D_{\mathcal{M}} \models_{N} \neg P(\bar{a})$. Since the analysis was done for an arbitrary value $\bar{a}$, $D_{\mathcal{M}}$ satisfies the constraint.

    - $\mathcal{M} \models_{N} (a_i \neq null)$. Hence $D_{\mathcal{M}}$ satisfies the constraint.

Lemma 5.4 still holds if IC also considers non-conflicting NNC. The proof does not need any modification. Since Lemmas 5.1, 5.2, 5.3 and 5.4 hold for sets of ICs containing NNCs, Propositions 5.4 and 5.5 also hold. Finally, given that these propositions hold, Theorem 5.5 also holds. $\square$

If we do not restrict ourselves to non-conflicting NNC, then the program would generate the Rep_d-repairs.

5.2.3 Head-cycle Free Programs

In some cases, the repair programs introduced in Section 5.2 can be transformed into equivalent non-disjunctive programs. This is the case when they become *head-cycle-free* [Ben-Eliyahu and Dechter, 1994]. Query evaluation from such programs has lower data complexity than general disjunctive programs, actually the data complexity is reduced from Π_2^P-complete to *coNP*-complete [Ben-Eliyahu and Dechter, 1994; Dantsin *et al.*, 1997]. We briefly recall their definition.

The *dependency graph* of a ground disjunctive program Π is the directed graph that has ground atoms as vertices, and an edge from atom A to atom B iff there is a rule with A (positive) in the body and B (positive) in the head. Π is *head-cycle free* (HCF) iff its dependency graph does not contain any directed cycles passing through two atoms in the head of the same rule. A disjunctive program Π is HCF if its ground version is HCF.

A HCF program Π can be transformed into a non-disjunctive normal program $sh(\Pi)$ that has the same stable models. It is obtained by replacing every disjunctive rule of the form $\bigvee_{i=1}^{n} P_i(\bar{x}_i) \leftarrow \bigwedge_{j=1}^{m} Q_j(\bar{y}_j), \varphi.$ by the n rules $P_i(\bar{x}_i) \leftarrow \bigwedge_{j=1}^{m} Q_j(\bar{y}_j), \varphi, \bigwedge_{k \neq i} not\ P_k(\bar{x}_k).$, for $i = 1, ..., n$.

For certain classes of queries and ICs, consistent query answering has a data complexity lower than Π_2^P, a sharp lower bound as seen in Theorem 5.3 (see also [Chomicki and Marcinkowski, 2005a]). In those cases, it is natural to consider this kind of transformations of the disjunctive repair program. In the rest of this section, we will consider sets *IC* of integrity constraints formed by UICs, RICs and NNCs.

Definition 5.13 A predicate P is *bilateral* with respect to *IC* if there exist ψ_1 and

ψ_2 in IC such that $P \in Ant(\psi_1)$ and $P \in Con(\psi_2)$. □

Note that ψ_1 and ψ_2 are not necessarily different.

Example 5.18 If $IC = \{\forall x(T(x) \rightarrow \exists y\ R(x,y), \forall xy(S(x,y) \rightarrow T(x))\}$, the only bilateral predicate is T. □

Theorem 5.6 For a set IC of UICs, RICs and NNCs, if for every $\psi \in IC$, it holds that (a) ψ has no bilateral predicates; or (b) ψ has exactly one occurrence of a bilateral predicate (without repetitions), then the program $\Pi(D, IC)$ is HCF. □

In order to prove Theorem 5.6, we need to introduce first the following lemma.

Lemma 5.5 For a set IC of UICs, RICs and NNCs, if there is a cycle in its dependency graph, then there exists at least one bilateral predicate. Furthermore, all the atoms in the cycle correspond to bilateral predicates.

Proof: First let us analyze which are the relationships between atoms in the dependency graph depending on the type of the constraints. First, note that the database atoms, aux, and atoms with constant $\mathbf{t}^{\star\star}$, will never be involved in a cycle, because they are exclusively in the head of rules (maybe negated in the body) or exclusively in the body. The only predicates that can be in a cycle are those with constants $\mathbf{t}^{\star}$, $\mathbf{t_a}$ and $\mathbf{f_a}$. We will concentrate on these atoms in what follows. The possible edges in the dependency graph between two different atoms in a UIC of the form (2.2) are shown in Figure 5.1. Figures 5.1(a) and 5.1(b) show the relationship between two predicates in the antecedent and consequent of IC respectively, and Figure 5.1(c) shows the relationship between a predicate in the antecedent and one in the consequent. For a RIC of the form (2.3) the edges are as in Figure 5.1(c). For a NNC, since there is unique database atom in it, the relationship is a simplified version of Figure 5.1(a)

(a) $P_i \in Ant(IC)$, $P_j \in Ant(IC)$

(b) $Q_n \in Con(IC)$, $Q_m \in Con(IC)$

(c) $P_i \in Ant(IC)$, $Q_n \in Con(IC)$
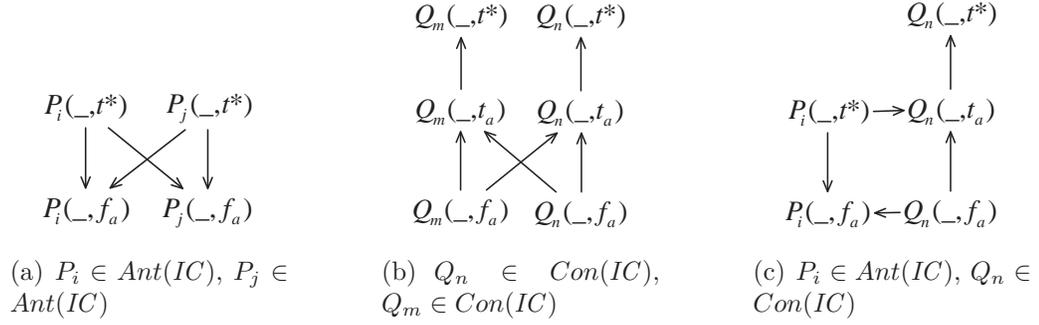
Figure 5.1: Possible dependency graphs of $\Pi(D, UIC)$

with only predicate P_i. It is clear from the figures that the only way we can have a cycle is by having a predicate in the consequent of a constraint (as a Q_m) and as an antecedent (as a P_i). It is easy to see that the predicates of all the atoms in the cycle will be bilateral $\qquad\square$

Proof of of Theorem 5.6: First, let us assume by contradiction that ψ has no bilateral predicates, but it is not HCF. This implies there is a cycle involving a pair of atoms in the head of a rule of $\Pi(D, IC)$. But, from Lemma 5.5 we know that if there is a cycle there is a bilateral predicate. We have reached a contradiction.

Now, let us assume by contradiction that ψ has exactly one occurrence of a bilateral predicate (without repetitions), but it is not HCF. This implies there is a cycle involving a pair of atoms in the head of a rule of $\Pi(D, IC)$. From Lemma 5.5 we know then that both atoms should be bilateral predicates. We have reached a contradiction. $\qquad\square$

As the following example shows, the conditions given in Theorem 5.6 are sufficient, but not necessary for the program to be HCF.

Example 5.19 If in IC we have the constraint $\forall xy(P(x,y) \rightarrow P(y,x))$, then P is a bilateral predicate, and the condition in the theorem is not satisfied. Therefore, the program $\Pi(D, IC)$ is not HCF. On the other hand, if we have instead

$IC' : \forall x(P(x,a) \rightarrow P(x,b))$, even though the condition is not satisfied, the program $\Pi(D, IC')$ is HCF. □

Example 5.20 If $IC = \{\forall xy(S(x,y) \rightarrow R(x)), \ \forall x(R(x) \rightarrow P(x))\}$, the only bilateral literal is R. Since there is no integrity constraint in IC that has more than one occurrence of R, the program $\Pi(D, IC)$ is HCF. As a consequence, we can replace, for example, the rule of $\Pi(D, IC)$:

$$S_{-}(x, y, \mathbf{f_a}) \vee R_{-}(x, \mathbf{t_a}) \leftarrow S_{-}(x, y, \mathbf{t}^{\star}), \ not \ R_{-}(x)$$

by the two rules

$$R_{-}(x, \mathbf{t_a}) \leftarrow S_{-}(x, y, \mathbf{t}^{\star}), \ not \ R_{-}(x), \ not \ S_{-}(x, y, \mathbf{f_a})$$

$$S_{-}(x, y, \mathbf{f_a}) \leftarrow S_{-}(x, y, \mathbf{t}^{\star}), \ not \ R_{-}(x), \ not \ R_{-}(x, \mathbf{t_a}) \qquad \square$$

Example 5.21 If $IC = \{\forall(S(x) \rightarrow R(x)), \ \forall x(P(x) \rightarrow S(x)), \ \forall xy(T(x,y) \rightarrow P(x))\}$, then the bilateral literals are S and P. Since the IC $\forall x(P(x) \rightarrow S(x))$ contains two bilateral literals we can not use Theorem 5.6 to determine if the program is HCF. □

Theorem 5.6 can be immediately applied to useful classes of ICs, like denial constraints, because they do not have any bilateral literals, and as a consequence, the repair program is HCF.

Corollary 5.1 If IC contains only ICs of the form $\forall \bar{x}(\bigwedge_{i=1}^{n} P_i(\bar{x}_i) \rightarrow \varphi)$, where $P_i(\bar{x}_i)$ is a database atom and φ is a formula containing built-in predicates only, then $\Pi(D, IC)$ is HCF. □

As a consequence of this corollary we obtain, for first-order queries and this class of ICs, that CQA belongs to *coNP*, because a query program (that is non-disjunctive) together with the repair program is still HCF. It has been shown that for this class

of constraints, if repairs are obtained only by tuple-deletion, the problem is *coNP-*complete [Chomicki and Marcinkowski, 2005a]. Actually, CQA for this class of ICs with our repair semantics is still *coNP*-complete, because the same reduction found in [Chomicki and Marcinkowski, 2005a] can be used in our case.

Example 5.22 For $IC = \{\forall xyzuv(P(x,y,z) \wedge P(x,u,v) \rightarrow y = u), \forall xyzuv(P(x, y, z) \wedge P(x,u,v) \rightarrow z = v), \forall xyzv(Q(x,y,z) \wedge P(x,y,v) \rightarrow z = v)\}$, and any ground instantiation, there are no bilateral literals. As a consequence, the program $\Pi(D, IC)$ will be always HCF. $\square$

5.3   Alternative Repair Semantics

Several alternative semantics have been considered in the literature. For example, repairs obtained by tuple updates of attributes [Wijsen, 2003; Wijsen, 2005; Bertossi *et al.*, 2005a; Bertossi *et al.*, 2005c; Bertossi *et al.*, 2005b; Bertossi *et al.*, 2005d], preference of insertion over deletion [Lembo *et al.*, 2002], tuple deletions [Chomicki and Marcinkowski, 2002; Chomicki and Marcinkowski, 2005b] and cardinality-based repairs [Arenas *et al.*, 2003; Bertossi and Chomicki, 2003; Lopatenko and Bravo, 2006; Lopatenko and Bertossi, 2006a; Lopatenko and Bertossi, 2006b]. Here, we would like to explore how we can modify the repair program in order to implement tuple deletions and cardinality-based repairs.

5.3.1   Tuple Deletion Repairs

Definition 5.4 defines the repairs obtained by tuple deletion. It is easy to modify the repair program in order to capture as stable models only those obtained by tuple deletions. The modified version of the program is as follows:

Definition 5.14 Given a database instance D, a set IC of UICs, RICs and NNCs, the repair program $\Pi^{del}(D, IC)$ contains the following rules:

1. Facts: $P(\bar{a})$ for each atom $P(\bar{a}) \in D$.

2. For every UIC ψ of the form (2.2), the rules:
$$\bigvee_{i=1}^{n} P_{i\text{-}}(\bar{x}_i, \mathbf{f_a}) \leftarrow \bigwedge_{i=1}^{n} P_i(\bar{x}_i), \bigwedge_{Q_j \in Q'} Q_{j\text{-}}(\bar{y}_j, \mathbf{f_a}), \bigwedge_{Q_k \in Q''} not\ Q_k(\bar{y}_k),$$
$$\bigwedge_{x_l \in \mathcal{A}(\psi)} x_l \neq null, \bar{\varphi}.$$
   for every set Q' and Q'' of atoms appearing in formula (2.2) such that $Q' \cup Q'' = \bigcup_{j=1}^{m} Q_j(\bar{y}_j)$ and $Q' \cap Q'' = \emptyset$. Here, $\mathcal{A}(\psi)$ 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.

3. For every RIC of the form (2.3), the rules:
$$P_{\text{-}}(\bar{x}, \mathbf{f_a}) \leftarrow P(\bar{x}),\ not\ \ aux(\bar{x}'), \bar{x}' \neq null.$$
$$aux(\bar{x}') \leftarrow Q(\bar{x}', \overline{null}),\ \ not\ Q_{\text{-}}(\bar{x}', \overline{null}, \mathbf{f_a}), \bar{x}' \neq null.$$
   For every $\ y_i \in \bar{y}$:
$$aux(\bar{x}') \leftarrow Q(\bar{x}', \bar{y}),\ \ not\ Q_{\text{-}}(\bar{x}', \bar{y}, \mathbf{f_a}), \bar{x}' \neq null, y_i \neq null.$$

4. For every NNC of the form (4.2), the rule:
$$P_{\text{-}}(\bar{x}, \mathbf{f_a}) \leftarrow P(\bar{x}), x_i = null.$$

5. For every predicate $P \in \mathcal{R}$, the interpretation rule:
$$P_{\text{-}}(\bar{x}, \mathbf{t^{\star\star}}) \leftarrow P(\bar{x}),\ \ not\ P_{\text{-}}(\bar{x}, \mathbf{f_a}). \qquad \square$$

This repair program can be obtained by deleting the atoms with annotated constant $\mathbf{t_a}$ from the rules that have this atom in the consequent, and by deleting completely the rules that have atoms annotated with $\mathbf{t_a}$ in the antecedent. Also, since the atoms with annotation constant $\mathbf{t^{\star}}$ will correspond exactly to the database facts, we can

replace all the atoms of the form $P_{\text{-}}(\bar{x}, \mathbf{t}^{\star})$ by $P(\bar{x})$. The resulting program computes the del-repairs for RIC-acyclic sets of UICs, RICs and non-conflicting NNC.

5.3.2 Minimum Cardinality

In [Dalal, 1992], in the context of belief revision, a semantics that minimizes the number of insertions and deletions was introduced. The repairs based in this intuition are called *cardinality-based repairs*. We will call *set-repairs* the repairs used so far (see Definition 5.3).

Repair programs to compute cardinality-based repairs for binary ICs. i.e. UICs with only two database atoms, are proposed in [Arenas *et al.*, 2003]. Those repair programs are like the programs to compute the *set-repairs* but with additional *weak constraints*. However, they do not deal with null values nor RICs. The results in [Arenas *et al.*, 2003] are based on the fact that every cardinality-based repair is also a set-repair

For databases with *null*, we can use the repair program given in Definition 5.12 together with some weak constraints to discard the models that do not have a minimal number of tuple insertions and deletions. We also consider a more general class of ICs than in [Arenas *et al.*, 2003], namely, ICs of the form (2.1) and (4.2).

Definition 5.15 Given three database instances D, D' and D'', D' is said to be closer to D than D'', denoted $D' \preceq_D^C D''$, iff (a) $|\Delta(D, D')| < |\Delta(D, D'')|$, or (b) $|\Delta(D, D')| = |\Delta(D, D'')|$ and $D' \leq_D D''$. □

So, a database D' is closer to database D than a database D'' if the number of tuples insertion and deletions required to transform D' into D are less than those required by D''. If they are equal, we prefer a database with *null*. The latter condition is to ensure that repairs are obtained using *null*.

Definition 5.16 Given a database instance D and a set IC of ICs of the form (2.1) and (4.2), a cardinality-based repair (C-repair) of D with respect to IC is a database instance D' over the same schema, such that $D' \models_N IC$ and D' is $\preceq^C_D$-*minimal* in the class of database instances that satisfy IC with respect to $\models_N$, and share the schema with D, i.e., there is no database D'' in this class with $D'' \prec^C_D D'$, where $D'' \prec^C_D D'$ means $D'' \preceq^C_D D'$ but not $D' \preceq^C_D D''$. The set of C-repairs of D with respect to IC is denoted with $Rep_C(D, IC)$. $\qquad\square$

Proposition 5.6 Given a database D with null values and a set IC of UICs, RICs and NNC, a C-repair of D with respect to IC is also a repair of D with respect to IC.

Proof: Let us assume by contradiction that there is a C-repair D' that is not a repair. Since D' is a C-repair, we know that $D' \models_N IC$. Also, since D' is not a repair, there should exist D'' such that $D'' \models_N IC$ and that $D'' \leq_D D'$ but not $D' \leq_D D''$. For $D' \leq_D D''$ not to be true there must exist $P(\bar{a}) \in \Delta(D, D')$ such that there does not exist an atom $P(\bar{a}')$, such that (a) $P(\bar{a}) \sqsubseteq P(\bar{a}')$, (b) $P(\bar{a}') \in \Delta(D, D'')$ and (c) if $P(\bar{a}) \sqsubset P(\bar{a}')$ then $P(\bar{a}') \notin \Delta(D, D')$. Also, since D' is a C-repair, there should also exist an atom $Q'(\bar{b}) \in \Delta(D, D'')$ such that there does not exist an atom $A \in \Delta(D, D')$ with $Q(\bar{b}) \sqsubseteq A$ (otherwise D'' would be a C-repair instead of D'). But his would imply that it is not true that $D' \leq_D D''$. We have reached a contradiction. $\qquad\square$

Since every C-repair is a repair, we could try to modify the repair program $\Pi(D, IC)$ by adding constraints to the disjunctive logic program to discard the stable models that do not lead to C-repairs. This can be done by using the so-called *weak constraints* [Leone *et al.*, 2006; Buccafurri *et al.*, 2000]. Weak constraints are program constraints that we would like to satisfy as much as possible, that is, we will choose the stable models that violate the weak constraints a smaller number of times. To distinguish

the traditional program constraints from weak constraints, we will express the latter with $\Leftarrow$ instead of $\leftarrow$.

In our case, we would like to minimize the number of deletions and insertions. In the repair program the number of deletions and insertions corresponds exactly to the number of atoms with annotated $\mathbf{t_a}$ and $\mathbf{f_a}$. Therefore, if we add weak constraints to avoid them, the stable model semantics for programs with weak constraints will choose only the models that minimize the number of atoms with $\mathbf{t_a}$ and $\mathbf{f_a}$.

Definition 5.17 Given a database instance D, a set IC of UICs, RICs and NNCs, the cardinality repair program $\Pi^C(D,\ IC)$ contains the following rules:

1. Rules of $\Pi(D, IC)$ (see Definition 5.12).

2. For every predicate P the weak constraints:

   $\Leftarrow P(x, \mathbf{t_a})$.

   $\Leftarrow P(x, \mathbf{f_a})$. $\hfill\square$

In [Leone *et al.*, 2006], different weights and priority levels can be assigned to each weak constraint. The models of a program with weak constraints minimize the sum of the weights of the violated weak constraints. If there are different priority levels, the semantics for weak constraints minimizes first the violation of the constraints at the highest priority level, and then it continues to the lower levels. Here, we want all of them to have the same weight and be at the same level. The weight (w) and level (l) can be specified after the constraint by $[w : l]$.

In our case we would have:

$\Leftarrow P(x, \mathbf{t_a})$. $[1 : 1]$

$\Leftarrow P(x, \mathbf{f_a})$. $[1 : 1]$

Example 5.23 Given a database $D = \{S(c, d), S(c, e)\}$ and the IC $\forall x(S(x, y) \rightarrow \exists R(x, z))$, there are two repairs, from which only one is a C-repair: $Rep(D, IC) =$

$\{\{S(c,d), S(c,e), R(c, null)\}, \emptyset\}$ and $Rep^C(D, IC) = \{\{S(c,d),\ S(c,e),\ R(c, null)\}\}$.

The program $\Pi^C(D, IC)$ is:

$$S(c,d). \qquad S(c,e).$$

$$S_{-}(x, y, \mathbf{f_a}) \vee R_{-}(x, null, \mathbf{t_a}) \leftarrow S_{-}(x, y, \mathbf{t^\star}),\ not\ aux(x), x \neq null.$$

$$aux(x) \leftarrow R(x, null),\ not\ R_{-}(x, null, \mathbf{f_a}), x \neq null.$$

$$aux(x) \leftarrow R_{-}(x, z, \mathbf{t^\star}),\ not\ R_{-}(x, z, \mathbf{f_a}), x \neq null, z \neq null.$$

$$R_{-}(x, y, \mathbf{t^\star}) \leftarrow R(x, y).$$

$$R_{-}(x, y, \mathbf{t^\star}) \leftarrow R_{-}(x, y, \mathbf{t_a}).$$

$$S_{-}(x, y, \mathbf{t^\star}) \leftarrow S(x, y).$$

$$S_{-}(x, y, \mathbf{t^\star}) \leftarrow S_{-}(x, y, \mathbf{t_a}).$$

$$R_{-}(x, y, \mathbf{t^{\star\star}}) \leftarrow R_{-}(x, y, \mathbf{t^\star}),\ not\ R_{-}(x, y, \mathbf{f_a}).$$

$$S_{-}(x, y, \mathbf{t^{\star\star}}) \leftarrow S_{-}(x, y, \mathbf{t^\star}),\ not\ S_{-}(x, y, \mathbf{f_a}).$$

$$\leftarrow R_{-}(x, y, \mathbf{t_a}), R_{-}(x, y, \mathbf{f_a}). \qquad \leftarrow S_{-}(x, y, \mathbf{t_a}), S_{-}(x, y, \mathbf{f_a}).$$

$\Pi(D, IC)$

$$\Leftarrow R_{-}(x, y, \mathbf{t_a}).\ \ [1:1]$$

$$\Leftarrow R_{-}(x, y, \mathbf{f_a}).\ \ [1:1]$$

$$\Leftarrow S_{-}(x, y, \mathbf{t_a}).\ \ [1:1]$$

$$\Leftarrow S_{-}(x, y, \mathbf{f_a}).\ \ [1:1]$$

Weak constraints

$\square$

It is easy to see that if we add the same weak constraints to program $\Pi^{del}(D, IC)$, we would get the repairs obtained by a minimal number of deletions.

5.4 Conclusions

In this chapter, we have provided mechanisms that would allow a user to specify, together with a query, a set of integrity constraints -that are not necessarily maintained by the DBMS- in such a way that the answers to the queries obtained from the system are consistent with the given semantic constraints.

We have introduced a new repair semantics that considers, systematically and for the first time, the possible occurrence of null values in a database in the form we find them present and treated in current commercial implementations. Null values are also used to restore the consistency of the database. The new semantics applies to a wide class of ICs, including cyclic sets of referential ICs.

We established the decidability of CQA under this semantics, and a tight lower and upper bound were obtained. The repairs under this semantics can be specified as stable models of a disjunctive logic program with a stable model semantics for acyclic foreign key constraints, universal ICs and *NOT NULL*-constraints, covering all the usual ICs found in database practice.

At the current state of this line of research, the methodology for computing CQA using repair programs provably works for any class of first-order ICs that contains RIC-acyclic universal constraints and referential constraints; in the sense that there is a one-to-one correspondence between repairs and stable models of the repair program.

The results in this chapter have been published in [Barceló *et al.*, 2003; Bravo and Bertossi, 2004; Bravo and Bertossi, 2006]. The semantics of CQA for RICs, an extension of the repair program, some optimization for it and some cases in which it is head-cycle free were introduced in [Barceló *et al.*, 2003; Bravo and Bertossi, 2004]. In both articles null values were used to repair the inconsistencies with respect to RICs, and tuples with null values where assumed never to produce inconsistencies (it did not matter if *null* was in relevant attributes or not). As a consequence, null values do not propagate in the repair process. The repair programs that considers the IC satisfaction semantics from Chapter 4 was presented in [Bravo and Bertossi, 2006].

Chapter 6

CQA in Data Integration Systems (DIS)

Independent and autonomous data sources can be virtually integrated by means of a mediator, which is a program that provides a global schema as an interface. The mediator is also responsible for generating query plans to answer global queries, by retrieving data sets from the sources, and combining them into a final answer set to be given back to the user.

The "Local-As-View" (LAV) approach to virtual data integration requires that each data source is described as a set of views over the global schema. On the other hand, the "Global-As-View" (GAV) approach, defines every global relation as a view of the set of relations in the sources (see [Lenzerini, 2002] for a survey of these and mixed approaches). Query answering is harder under LAV [Abiteboul and Duschka, 1998]. On the other hand, LAV offers more flexibility to accept or release sources into/from an existing system.

In this virtual integration setting, inconsistencies with respect to global integrity constraints (ICs), i.e., that refer to the relations at the virtual level, are likely to occur. This is due to the autonomy of the participating sources, the lack of a central maintenance mechanism; and also to the flexibility to add or delete sources, without having to consider the other sources in the system.

Example 6.1 Consider the LAV based global integration system $\mathcal{G}_1$ with a global relation $R(X, Y)$ and two source relations $v_1 = \{V_1(a,\ b),\ V_1(c,\ d)\}$ and $v_2 = \{V_2(a,\ c), V_2(d,\ e)\}$ that are described by the view definitions $V_1(x, y) \leftarrow R(x,\ y);\ V_2(x, y)$

$\leftarrow R(x, y)$. The global functional dependency (FD) $R \colon X \to Y$ is violated through the pair of tuples $\{(a, b), (a, c)\}$. □

In a DIS there is an intuitive notion of consistent answer to a query.

Example 6.2 (example 6.1 continued) If we pose on the global system the query $Q \colon Ans(x, y) \leftarrow R(x, y)$, we obtain the answers $\{Ans(a, b), Ans(c, d), Ans(a, c), Ans(d, e)\}$. However, only the tuples $Ans(c, d), Ans(d, e)$ should be returned as consistent answers with respect to the FD $R(X, Y) \colon X \to Y$. □

Several algorithms for deriving query plans to obtain query answers from virtual data integration systems have been proposed in the last few years (see [Levy, 2000] for a survey). However they are not designed to obtain consistent answers to queries. Furthermore, some of those algorithms assume that certain ICs hold at the global level [Gryz, 1999; Duschka *et al.*, 2000; Grant and Minker, 2002]. This may not be a realistic assumption due to the independence of the different data sources and the lack of a central, global integrity maintenance mechanism. Only a few papers consider the problem of CQA in virtual integration systems [Lembo *et al.*, 2002; Bertossi *et al.*, 2002; Bravo and Bertossi, 2003; Calì *et al.*, 2003b].

In a virtual data integration system, the mediator should solve potential inconsistencies when the query plan is generated; again without attempting to bring the whole system into a global consistent material state. Such an enhanced query plan generator should produce query plans that are guaranteed to retrieve all and only the consistent answers to global queries.

In this spirit and under the LAV approach, in [Bertossi *et al.*, 2002] a methodology for generating query plans to compute answers to limited forms of queries that are consistent with respect to a restricted class of universal ICs was presented. This methodology uses the query rewriting approach to CQA presented in [Arenas *et al.*,

1999]; and as a consequence inherits its limitations in terms of the queries and ICs that it can handle. Once the query is transformed, query plans are generated for the new query. In [Bertossi *et al.*, 2002], the authors also provides a semantics for CQA in mediated integrated systems (see Section 6.1).

In this chapter, we address the problem of retrieving consistent answers to global queries under the LAV approach, and assuming that sources are open (or incomplete or sound) [Abiteboul and Duschka, 1998]. We consider arbitrary universal ICs and referential ICs; that is, the ICs that are used most in database practice [Abiteboul *et al.*, 1995]. View definitions are conjunctive queries, and disjunctions thereof. Global queries are expressed in Datalog and its extensions with negation.

The methodology can be summarized as follows. In a first stage, we specify, using a logic program with the *choice operator* [Giannotti *et al.*, 1997] and stable model semantics [Gelfond and Lifschitz, 1991], the class of all minimal legal global instances of a virtual integration system. This approach is inspired by the inverse-rules algorithm [Duschka *et al.*, 2000] and uses auxiliary Skolem predicates whose functionality is enforced with the choice operator.

In order to obtain answers to global queries from the DIS, a query program has to be combined with the program that specifies the minimal instances, and then be run under the skeptical stable model semantics. It turns out that *minimal answers*, i.e., answers that are true in all minimal instances, can be retrieved for $Datalog^{not}$ queries. The *certain answers*, i.e., those true in all legal global instances, can be obtained for all monotone queries, a result that generalizes those found so far in the literature.

In a second stage, we address the computation of consistent answers. We first observe that an integration system is consistent if all of its minimal legal instances satisfy the integrity constraints [Bertossi *et al.*, 2002]. Consistent answers from an inconsistent integration systems are those that can be obtained from all the repairs, as

described in Definition 5.3, of all the minimal legal instances with respect to the global ICs [Arenas *et al.*, 1999; Bertossi *et al.*, 2002]. Note that *null* may be present in the database sources and that it will also be used to repair inconsistencies with respect to RICs. In order to retrieve consistent answers, the specification of the minimal instances has to be combined with a specification of their repairs with respect to the given ICs. The latter is a logic program that specifies the repairs as its stable models; and uses annotation constants as in the case of repairs of single relational databases [Arenas *et al.*, 1999], as presented in Chapter 5. We have experimented with this query answering mechanism (and the computation of minimal instances and their repairs) with the *DLV* system [Eiter *et al.*, 2000; Leone *et al.*, 2006], which implements the stable model and answer set semantics of disjunctive extended deductive databases.

This chapter is structured as follows. In Section 6.1, we review some basic notions we need in the rest of the chapter. In Section 6.2, the minimal legal global instances of a mediated system are specified by means of logic programs with a stable model semantics. In Section 6.3, the repairs of the minimal global instances are specified as the stable models of disjunctive logic programs, like those used to specify repairs of single relational databases for CQA in Section 5.2. In Section 6.4, consistent answers to queries are obtained by running a query program in combination with the previous two specification programs. In Section 6.5, several issues and possible extensions around the specification presented in the previous sections are discussed in detail. Finally, in Section 6.6, we draw some final conclusions, and we point to related and future work.

6.1 Preliminaries

6.1.1 Data Integration System (DIS)

A DIS uses a mediator between the user and the data sources. The main features of a mediator based system are: (a) The interaction with the system via queries posed to the mediator; (b) Updates via the mediator are not allowed; (c) Data sources are mutually independent and may participate in different mediated systems at the same time; (d) Sources are allowed connect and disconnect from the integration system; (e) Data is kept in the local, individual sources, and extracted at the mediator's request.

General Architecture of an Integration System
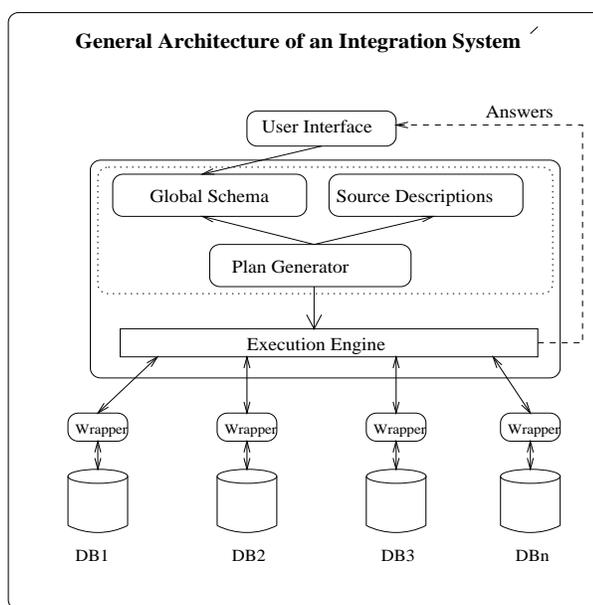
Figure 6.1: Architecture of an integration system

The mediator interacts with the users or applications as if it was a single database. In order to achieve this, it provides a *global schema*, consisting of a set of names for relations (virtual tables) and their attributes. This schema is application dependent and determines a (family of) query language(s), as in a usual relational databases from the user's point of view. However, the database instances corresponding to the

global schema is virtual.

A user poses queries to the mediator in terms of the relations in the global schema. However, in order to answer those global queries, the mediator needs to knows the correspondence between the global schema and the local schemas. This is achieved by means of a set of source descriptions, i.e., descriptions of what data can be found in the different sources. Having this information, when the mediator receives a query, it develops a *query plan* that determines: (a) the portions of data that are relevant to the query at hand, (b) their locations in the relevant data sources, (c) how to extract that data from the sources via queries, and (d) how to combine the answers received into a final answer for the user. Figure 6.1 shows the main elements in the architecture of a mediator for virtual integration of data sources.

The mediator is responsible for solving problems of redundancy, incompleteness, and consistency of data in the integration system. In this chapter, we will consider the latter problem, a very relevant one in this context. For example, what should the mediator do if it is asked about a person's ID card number and it gets two different numbers, each coming from a different source? The two sources, taken independently and separately, may be consistent, but taken together, possibly not. Such consistency problems are likely and natural in virtual data integration. Notice that consistency problems in virtual integration, unlike the "materialized" approaches to data integration, which offer data reconciliation solutions, cannot be solved a priori, at the physical data level.

Another element shown in Figure 6.1 is the *wrapper*. This is a module that is responsible for wrapping a data source in such a way that the latter can interact with the rest of integration system. It provides the mediator with data from a source as requested by the execution engine. As a consequence, it presents a data source as a database, with schema and data format that can be understood and used by the

mediator. Notice that this presentation schema may be different the data source. Actually, it may be the case that the source is not at all internally structured as a database, but this should be transparent to the mediator. All this may require preliminary transformations, cleaning, etc., before the data can be exported to the integration system. There is one or more wrappers for each data source. In the following, we will assume that each data source already has a wrapper that presents it as a relational database.

Example 6.3 Consider a global schema for a database "containing" information about music albums: $CD(Album, Artist, Year)$, $Contract(Artist, Year, Label)$, $Songs(Album, Song)$. Now, a user wants to know the name of the label with which Norah Jones had a contract during 2002. This is asked issuing the following query to the global system Q: $Ans(l) \leftarrow Contract(NorahJones, 2002, l)$.

Here, the predicate Ans will contain the answers, that are to be computed using the expression on the RHS of this rule. In this case, this is a simple relational selection followed by a projection: $\Pi_{Label}\sigma_{Artist="Norah\ Jones" \wedge Year=2002} Contract$.

It is a problem that the data is not in the virtual global relation $Contract$, but in the data sources $DB_1(Album, Artist, Year)$, $DB_2(Album, Artist, Year, Label)$, $DB_3(Album, Song)$. As a consequence, a query plan is needed in order to extract and combine the relevant data from the data sources. However, in order to design such a plan, the mediator needs to know the correspondence between the virtual global relations and the data sources. □

A key element in the mediator architecture is the set of *source descriptions*, i.e., the descriptions of the available sources and their contents (as presented by the wrapper), which is achieved by establishing the relationships (mappings) between the global schema and the local schemata. These descriptions are given by means of a

set of logical formulas; similar to views definitions in terms of base tables in a relational database, i.e., using queries written in a query language. Usually those query languages use logical formulas or their SQL versions.

With respect to how mappings are defined, there are two main approaches (and combinations of them): (a) *Global-as-View* (GAV), under which the relations in the global schema are described as views of the collection of local relations [Ullman, 2000]; and (b) *Local-as-View* (LAV), under which each relation in a local source is described as a view of the global schema [Levy *et al.*, 1996]. GLAV denotes a combination of GAV and LAV [Friedman *et al.*, 1999] where the rules can have more than one atom in the head. Another approach, called *Both-as-View* (BAV), consists of a specification of the transformation of the local schema into the given global schema, in such a way that each schema can be defined in terms of the other schema [McBrien and Poulovassilis, 2003]. In Section 6.1.2, we describe and compare the GAV and LAV approaches.

The *plan generator* gets a user query in terms of global relations and uses the source descriptions to design a *query plan*. This is achieved by *rewriting* the original query as a set of subqueries that are expressed in terms of the local relations. The query plan includes prescriptions on how the answers from the local sources have to be combined. The query rewriting process executed by the plan generator strongly depends on whether the LAV or the GAV approach is followed. Still much theoretical and technical research is going on in relation to query plan generation. The plan is executed by the *execution engine*. Notice that it should be the plan generator who takes care of anticipating and solving potential inconsistencies. It should solve them in advance, when the plan is being generated. Later in this chapter, we will explore this issue in detail.

6.1.2 Description of Data Sources

The global/local schema mappings or, equivalently, the descriptions of the source contents are expressed through logical formulas that relate the global and local relations.

Global-as-View

In this case, the relations in the global schema are described as *views* over the tables in the union of the local schemata. This is conceptually very natural, because views are usually virtual relations defined in terms of relations (tables); and here we have global relations that are virtual, and local sources that are materialized. The views are described Datalog notation [Halevy, 2001; Lenzerini, 2002].

Example 6.4 (example 6.3 continued) Assume the relation CD is defined using the views

$$CD(Album, Artist, Year) \leftarrow DB_1(Album, Artist, Year),$$

$$CD(Album, Artist, Year) \leftarrow DB_2(Album, Artist, Year, Label).$$

Relation CD is defined as the union of the projections of DB_1 and DB_2 on attributes $Album, Artist, Year$, i.e., in relational terms, defined by

$$CD \supseteq \Pi_{Album, Artist, Year}(DB_1) \cup \Pi_{Album, Artist, Year}(DB_2).$$

The global relation *Songs* and *Label* are defined by:

$$Songs(Album, Song) \leftarrow DB_1(Album, Artist, Year), DB_3(Album, Song).$$

$$Contract(Artist, Year, Label) \leftarrow DB_2(Album, Artist, Year, Label).$$

The first view is defined as the join of DB_1 and DB_3 via attribute $Album$, and with a

projection on *Album, Song*. The second view is defined as the projection of DB_2 on *Artist, Year, Label*.

These views have been defined by means of *rules*. Each rule specifies that, in order to compute the tuples in the relation in the LHS (the *head* of the rule), one has to go to the RHS (the *body* of the rule) and compute whatever is specified there. The attributes appearing in the head indicate that they are the attributes of interest, thus the others (in the body) can be projected out at the end. If there are more that one rule to compute a single relation, we use all of them and we take the union of the results, as for the relation *CD*.

Instead of using a rule as above, we could have used relational algebra (or relational calculus, or SQL), in the case of the relation *Songs*:

$$Songs = \Pi_{Album,Song}(DB_1 \bowtie_{Album} DB_3). \qquad \Box$$

Once the global relations have been defined as views, we may start posing global queries, i.e., queries expressed in terms of the global relations. The problem is to answer them considering that the global relations do not contain data. Under the GAV approach this is simple, all we need to do is *rule unfolding* [Halevy, 2001].

Example 6.5 (example 6.4 continued) Consider the following global query about the music albums released in the year 2003, with their artists and songs

$$Ans(Album, Artist, Song) \leftarrow \underline{CD(Album, Artist, 2003)}, \underline{\underline{Songs(Album, Song)}}.$$

Since the query is expressed in terms of the global schema, the data has to be obtained from the sources, that is, the query has to be *rewritten* in terms of the source relations. We do this by unfolding each global relation, replacing it by its definition in terms of the local relations. We have underlined differently the goals in the body in order to

keep track of the rewriting for each of them.

$$Ans'(Album, Artist, Song) \quad \leftarrow \quad \underline{\underline{DB_1(Album, Artist, 2003)}},$$

$$\underline{\underline{DB_1(Album, Artist2, Year), DB_3(Album, Song)}}.$$

$$Ans'(Album, Artist, Song) \quad \leftarrow \quad \underline{\underline{DB_2(Album, Artist, 2003, Label)}},$$

$$\underline{\underline{DB_1(Album, Artist2, Year), DB_3(Album, Song)}}.$$

These new queries do get answers directly from the sources; and the final answer is the union of two answer sets, one for each of the rules. $\square$

If, in addition to the view definitions, there might be global ICs that have to be satisfied by the system. Simple unfolding, in the form illustrated above, is not enough for query answering if ICs have to be respected [Calì *et al.*, 2002a; Calì *et al.*, 2002c].

Local-as-View

Under the LAV approach, each table in each local data source is described as a view (i.e. as a query expression) in terms of the global relations [Ullman, 2000; Lenzerini, 2002]. This may seem somehow unnatural or unusual from a conceptual point of view, and from perspective of databases practice, because here the views contain the data, but not the "base tables". However, as we will see, this approach has some advantages.

More precisely, in the general situation we have a collection of data sources (think of a collection of relational tables) $S_1, \ldots, S_n$, and a global schema G for the system that integrates data from $S_1, \ldots, S_n$. Tables in $S_1, \ldots, S_n$ are seen as *views* over G, and as a consequence, they can be defined by query expressions over the global schema.

Example 6.6 Consider the sources S_1, S_2 that are defined by the view expressions

$$S_1: \quad V_1(Album, Artist, Year) \leftarrow \quad CD(Album, Artist, Year),$$

$$Contract(Artist, Year, emi), Year \geq 1990.$$

$$S_2: \quad V_2(Album, Song) \leftarrow \quad Songs(Album, Song).$$

Source S_1 contains a table whose entries are albums produced after 1990 by the label EMI with their artists and years. Source S_2 contains one table with songs and their albums.

Those relations that are not defined as views belong to the global schema G, in this case, they are the relations: $CD(Album, Artist, Year)$, $Songs(Album, Song)$, $Contract(Artist, Year, Label)$. $\square$

Notice that from the perspective of S_1, there could be other sources containing information about albums produced by EMI after 1990, and that complementary information could be exported into the global system. In this sense, the information in S_1 could be considered as "incomplete" with respect to what G contains (or might contain). In other words, S_1 contains only a part of the data of the same kind in the global system. We will elaborate on this later on. Finally, also notice that in the previous example, and this is a general situation under LAV, the definition of each source does not depend on other sources.

Now we want to answer global queries under LAV.

Example 6.7 (example 6.6 continued) The following query posed on G asks for the songs with its album and the year they were released:

$$Ans(Album, Song, Year) \leftarrow CD(Album, Artist, Year), Songs(Album, Song).$$

This query is expressed as usual, in terms of global relations only, however, it is not possible to obtain the answers by a simple and direct computation of the RHS of the query. Now, there is no direct rule unfolding mechanism for the relations in the body, because we do not have explicit definitions for them. And the data resides in the sources, which are now defined as views.

We can see that plan generation to extract information from the sources becomes more complex under LAV than under GAV. Since a query plan is a rewriting of the query as a set of queries on the sources and a prescription on how to combine their answers (as is needed in this example), the following could be a query plan to answer the original query:

$$Ans'(Album, Song, Year) \leftarrow V_1(Album, Artist, Year), V_2(Album, Song).$$

The query has been rewritten in terms of the views; and in order to obtain the final answer, we first extract values for *Album*, *Year* from V_1; then we extract the tuples from V_2; finally, at the mediator level, we compute the join via *Album*.

Notice that due to the limited contents of the sources, we only obtain albums produced by EMI after 1990. □

In LAV, as in GAV, we pose a query in terms of the global relations, but we have to answer using the contents of certain views only (the local relations). As a consequence, under LAV query plan generation becomes an instance of a more general and traditional problem in databases, the one of *query rewriting using views*.

To see this connection more clearly, assume we have a collection of views $V_1, \ldots, V_n$, whose contents have already been computed, and cached or materialized. When a new query Q arrives, instead of computing its answers directly, we try to use the answers (contents) of $V_1, \ldots, V_n$. A problem to consider consists of determining how

much we get from the real answer by using the pre-computed views only; and also determining what is the maximum we can get in terms of the kind of views we have available. The research carried out in query answering using views [Levy *et al.*, 1995; Abiteboul and Duschka, 1998; Gupta and Mumick, 1999; Halevy, 2001; Halevy, 2000; Flesca and Greco, 2001] and query containment [Abiteboul and Duschka, 1998; Kolaitis and Vardi, 2000; Millstein *et al.*, 2003; Calvanese *et al.*, 2003] has become quite relevant to the area of data integration.

6.1.3   Comparison of Paradigms

We have seen that under GAV, rule unfolding makes plan generation simple and direct. For the LAV approach, plan generation is provably more difficult [Abiteboul and Duschka, 1998; Lenzerini, 2002; Calì *et al.*, 2002b; Ullman, 2000].

In terms of flexibility to adding and deleting sources into/from the system, GAV mappings are in general less flexible than LAV mappings. If GAV mappings combine data of different sources, adding or deleting sources might imply modifying the definitions of the global relations. LAV offers more flexibility to add new sources or delete old ones, because a new source is just a new view definition. On the other hand, if the GAV mappings describe each global relations in terms of only one source, both GAV and LAV have the same level of flexibility.

It is interesting to know that given a set of sources and a global schema, it is in general not possible to find a LAV and a GAV mapping that provide the same semantics to the system.

Example 6.8 (example 6.4 continued) If we try to use a LAV mapping for this sources and global schema, we would need to describe DB_1, DB_2 and DB_3 in terms

of *Songs*, *CD* and *Contract*. A possible mapping under the LAV approach is:

$$DB_1(Album, Artist, Year) \leftarrow CD(Album, Artist, Year).$$

$$DB_2(Album, Artist, Year) \leftarrow CD(Album, Artist, Year), Contract(Artist, Year, Label).$$

$$DB_3(Album, Song) \leftarrow Songs(Album, Song).$$

This mappings give a different meaning to the DIS. For example, for the GAV mapping, it is possible to have a tuple in table $Songs(Album, Song)$ that is not present for the LAV mapping. This can happen if there is an *Album* in DB_3 that is not in DB_1. $\qquad\qquad\square$

In this sense, the decision of which of the two approaches to use, it is not necessarily related to flexibility or how simple the query plan generation is, but to which approach gives a better description of the intended meaning of the DIS.

For more comparisons between LAV and GAV refer to [Levy, 2000; Ullman, 2000; Lenzerini, 2002]. An approach that generalizes both LAV and GAV mappings is GLAV [Friedman *et al.*, 1999]. The results presented in this Chapter can be used in conjunction for the GLAV approach.

The flexibility to add/remove sources, in particular under LAV, is likely to introduce extra sources of inconsistencies which will have to be dealt with. This is why we will first concentrate on the LAV approach in the rest of this chapter. In Section 6.5.6, we will address the problem of CQA for the GAV approach.

6.1.4 Global Schemas and View Definitions

A *global schema* $\mathcal{R}$ consists of a finite set of relations $\{R_1, R_2, ..., R_m\}$ over a fixed, possibly infinite domain $\mathcal{U}$ that may not contain *null*. With these relation symbols and the elements of $\mathcal{U}$ treated as constants, a first-order language $\mathcal{L}(\mathcal{R})$ can be defined.

This language can be extended with defined and built-in predicates, like (in)equality. In particular, we will extend the global schema with a *local schema* $\mathcal{S}$, i.e., a finite set of new view predicates $V_1, V_2, ..., V_n$, that will be used to describe the relations in the local sources.

A *view*, denoted by a new predicate V, is defined by means of conjunctive query [Abiteboul *et al.*, 1995], i.e., an $\mathcal{L}(\mathcal{R} \cup \mathcal{S})$-formula φ_V of the form $V(\bar{t}) \leftarrow body(\varphi_V)$, where $\bar{t}$ is a tuple containing variables and/or constants. For the LAV approach $body(\varphi_V)$ is a conjunction of $\mathcal{R}$-atoms, and $V \in \mathcal{S}$.

Given a database instance D over schema $\mathcal{R}$, and a view definition φ_V, $\varphi_V(D)$ denotes the extension of V obtained by applying the definition φ_V to D. If the view already has an extension v (corresponding to the contents of a data source), it is possible that v is incomplete and stores only some of the tuples in $\varphi_V(D)$; i.e. $v \subseteq \varphi_V(D)$, and we say the view extension v is *open* with respect to D [Abiteboul and Duschka, 1998]. Most mechanisms for deriving query plans assume that sources are open, e.g. [Duschka *et al.*, 2000].

A *source S* is a pair $\langle \varphi, v \rangle$, where φ is the view definition, and v is an extension for the view defined by φ. An *open global system* $\mathcal{G}$ is a finite set of open sources. The global schema $\mathcal{R}$ consists of the relation names that do not have a definition in the global system. The underlying domain $\mathcal{U}$ for $\mathcal{R}$ is a possibly proper superset of the *active domain*, which consists of all the constants appearing in the view extensions v_i of the sources, and in their definitions. When considering global integrity constraints the *active domain* also includes the constants in them. A global system $\mathcal{G}$ defines a set of legal global instances (see [Lenzerini, 2002] for a survey of notions in virtual data integration).

Definition 6.1 Given an open global system $\mathcal{G} = \{\langle \varphi_1, v_1 \rangle, \ldots, \langle \varphi_n, v_n \rangle\}$, the set of legal global instances is $Linst(\mathcal{G}) = \{D \text{ instance over } \mathcal{R} \mid v_i \subseteq \varphi_i(D), i = 1, \ldots, n\}$.

□

Example 6.9 (example 6.2 continued) Let us denote by φ_1, φ_2 the view definitions of V_1, V_2, resp. in $\mathcal{G}_1$. The database $D = \{R(a, b), R(c, d), R(a, c), R(d, e)\}$ is a legal global instance, because $v_1 = \{V_1(a, b), V_1(c, d)\} \subseteq \varphi_1(D) = \{V_1(a, b), V_1(c, d),$ $V_1(a, c), V_1(d, e)\}$, and $v_2 = \{V_2(a, c), V_2(d, e)\} \subseteq \varphi_2(D) = \{V_2(a, b), V_2(c, d), V_2(a,$ $c), V_2(d, e)\}$. Supersets of D are also legal instances; but proper subsets are not. □

Example 6.10 Let $\mathcal{U} = \{a, b, c, \ldots\}$ be the underlying domain. Consider the integration system $\mathcal{G}_2$ defined by

$$V_1(x, z) \leftarrow P(x, y), R(y, z); \qquad v_1 = \{(a, b)\}.$$
$$V_2(x, y) \leftarrow P(x, y); \qquad\qquad v_2 = \{(a, c)\}.$$

Each global instance D of the form $\{P(a, c), P(a, z), R(z, b)\}$, with $z \in \mathcal{U}$, is a legal instance, because $v_1 \subseteq \varphi_1(D) = \{(a, b)\}$ and $v_2 \subseteq \varphi_2(D) = \{(a, c), (a, z)\}$. Any superset of D is also legal, but none of its subsets is. □

The semantics of query answers in mediated integration systems is given by the notion of *certain answer*. In this chapter, we will consider queries expressed in Datalog and its extensions with negation.

Definition 6.2 [Abiteboul and Duschka, 1998] Given an open global system $\mathcal{G}$ and a global query $Q(\bar{x}) \in \mathcal{L}(\mathcal{R})$, a ground tuple $\bar{t}$ is a *certain answer* to Q in $\mathcal{G}$ if for every global instance $D \in Linst(\mathcal{G})$, it holds that $D \models Q[\bar{t}]$.[1] We denote with $Certain_{\mathcal{G}}(Q)$ the set of certain answers to Q in $\mathcal{G}$. □

Example 6.11 (example 6.9 continued) Consider the following global query Q posed on $\mathcal{G}_1$: $Ans(x, y) \leftarrow R(x, y)$. In this case, $Certain_{\mathcal{G}_1}(Q) = \{(a, b), (c, d), (a, c), (d, e)\}$. □

---

[1] $D \models Q[\bar{t}]$ means that query $Q(\bar{x})$ is true in instance D, when the tuple of variables $\bar{x}$ is assigned the values in the tuple $\bar{t}$ of database elements.

The inverse-rules algorithm [Duschka *et al.*, 2000] for generating query plans under the LAV approach assumes that sources are open and each source relation V is defined as a conjunctive view over the global schema: $V(\bar{x}) \leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$, with $\bar{x} \subseteq \bigcup_i \bar{x}_i$. Since the queries posed on the system are expressed in terms of the global relations, that now appear in the bodies of the view definitions (contrary to the GAV approach), those definitions cannot be applied directly . The rules need to be "inverted".

For $j = 1, \ldots n$, $P_j(\bar{x}'_j) \leftarrow V(\bar{x})$ is an "inverse rule" for P_j. The tuple $\bar{x}_j$ is transformed to obtain the tuple $\bar{x}'_j$ as follows: if $x \in \bar{x}_j$ is a constant or is a variable appearing in $\bar{x}$, then x is unchanged in $\bar{x}'_j$. Otherwise, x is a variable x_i that does not appear in $\bar{x}$, and it is replaced by the term $f_i(\bar{x})$, where f_i is a fresh Skolem function. We denote the set of inverse rules of the collection $\mathcal{V}$ of source descriptions in $\mathcal{G}$ by $\mathcal{V}^{-1}$.

Example 6.12 (example 6.10 continued) The set $\mathcal{V}^{-1}$ consists of the rules $P(x, f(x,z)) \leftarrow V_1(x,z)$; $R(f(x,z),z) \leftarrow V_1(x,z)$; and $P(x,y) \leftarrow V_2(x,y)$. For a view definition, we need as many Skolem functions as existential variables in it. For example, if instead of $V_1(x,z) \leftarrow P(x,y), R(y,z)$ we had, say $V_1(x,z) \leftarrow P(x,y), R(y,z,w)$, we would need two Skolem functions for that view, and the inverse rules arising from that view would be $P(x, f(x,z)) \leftarrow V_1(x,z)$ and $R(f(x,z),z,g(x,z)) \leftarrow V_1(x,z)$. $\square$

The inverse rules are then used to answer Datalog queries expressed in terms of the global relations, that now, through the inverse rules, have definitions in terms of the sources. The query plan obtained with the inverse rule algorithm is maximally contained in the query [Duschka *et al.*, 2000], and the answers it produces coincide with the certain answers [Abiteboul and Duschka, 1998].

6.1.5 Consistency

We assume that we have a set of global integrity constraints $IC \subseteq \mathcal{L}(\mathcal{R})$ containing UICs and RICs. The results here can be easily extended to also consider NNCs.

Definition 6.3 [Bertossi *et al.*, 2002] (a) Given a global system $\mathcal{G}$, an instance D is *minimal* if $D \in Linst(\mathcal{G})$ and is minimal with respect to set inclusion, i.e., there is no other instance in $Linst(\mathcal{G})$ that is a proper subset of D (as a set of ground atoms). We denote by $Mininst(\mathcal{G})$ the set of minimal legal global instances of $\mathcal{G}$ with respect to set inclusion. (b) A global system $\mathcal{G}$ is *consistent* with respect to IC, if for all $D \in Mininst(\mathcal{G})$, $D \models IC$. □

Example 6.13 (example 6.12 continued) Assume that $\mathcal{G}_2$ has the source contents $v_1 = \{V_1(a,b)\}$, $v_2 = \{V_2(a,c)\}$, and that $\mathcal{U} = \{a,b,c,u,...\}$. Then, the elements of $Mininst(\mathcal{G}_2)$ are of the form $D_z = \{P(a,z), R(z,b), P(a,c)\}$, for some $z \in \mathcal{U}$. The global FD $P(X,Y)\colon X \to Y$ is violated exactly in those minimal legal instances D_z for which $z \neq c$. Thus, $\mathcal{G}_2$ is inconsistent. □

Example 6.14 (examples 6.1 and 6.2 continued) This DIS has only one minimal legal instance: $\{R(a,b), R(c,d), R(a,c), R(d,e)\}$. Since this instance does not satisfy the FD $R(X,Y)\colon X \to Y$, the system is inconsistent. □

Definition 6.4 [Bertossi *et al.*, 2002] Given an open global system $\mathcal{G}$ and a global query $Q(\bar{x}) \in \mathcal{L}(\mathcal{R})$, a ground tuple $\bar{t}$ is a *minimal answer* to Q in $\mathcal{G}$ if for every global instance $D \in Mininst(\mathcal{G})$, it holds $D \models Q[\bar{t}]$. We denote with $Minimal_{\mathcal{G}}(Q)$ the set of minimal answers to Q in $\mathcal{G}$. □

Clearly $Certain_{\mathcal{G}}(Q) \subseteq Minimal_{\mathcal{G}}(Q)$. For monotone queries [Abiteboul *et al.*, 1995], the two notions coincide [Bertossi *et al.*, 2002]. Nevertheless, in Example 6.13 the query $Ans(x,y) \leftarrow$ *not* $P(x,y)$ has (b,a) as a minimal answer, but not as a certain

answer, because there are legal instances that contain $P(b, a)$. Since consistency was defined with respect to minimal global instances, the notion of minimal answer is particularly relevant.

Definition 6.5 [Arenas *et al.*, 1999] Let D, D' be database instances over the same schema and domain. The *distance*, $\Delta(D, D')$, between D and D' is the symmetric difference $\Delta(D, D') = (\Sigma(D) \smallsetminus \Sigma(D')) \cup (\Sigma(D') \smallsetminus \Sigma(D))$. □

We will assume that if the database has null values we will consider the definition of IC satisfaction as defined in Section 4.1.

Example 6.15 Consider the universal IC $\forall xy(P(x, y) \rightarrow R(x, y))$ and the referential IC $\forall x(T(x) \rightarrow \exists y P(x, y))$. The database instance $D = \{P(a, d),\ R(a, d),\ T(a),\ T(b),\ P(b, null)\}$ is consistent. The universal constraint is satisfied even in the presence of $P(b, null)$, since the incomplete information in relevant attributes cannot generate inconsistencies. □

Definition 6.6 (based on [Arenas *et al.*, 1999]) Let $\mathcal{G}$ be a global system and IC a set of global ICs. A *repair* of $\mathcal{G}$ with respect to IC is a global database instance D', such that $D' \models_N IC$ and D' is $\leq_D$-minimal for some $D \in Mininst(\mathcal{G})$. □

The semantics of satisfaction of ICs is the one introduced in Section 4.1.

Example 6.16 Consider the UIC $\forall xy(P(x, y) \rightarrow R(x, y))$, together with the RIC $\forall x(T(x) \rightarrow \exists y P(x, y))$, and the inconsistent minimal legal instance $D = \{P(a, b), T(c)\}$. The repairs for the latter are:

i	D_i	$\Delta(D, D_i)$
1	$\{P(a, b), R(a, b), T(c), P(c, null)\}$	$\{R(a, b), P(c, null)\}$
2	$\{P(a, b), R(a, b)\}$	$\{T(c), R(a, b)\}$
3	$\{T(c), P(c, null)\}$	$\{P(a, b), P(c, null)\}$
4	$\emptyset$	$\{P(a, b), T(c)\}$

We also have that the instance $D_5 = \{P(a,b), R(a,b), T(c), P(c,a)\}$, where we have introduced $P(c,a)$ in order to satisfy the referential IC, does satisfy IC, but is not a repair because $\Delta(D, D_1) \leq_D \Delta(D, D_7) = \{R(a,b), P(c,a)\}$. $\qquad\square$

We can see that a repair of a global system is a global database instance that satisfies IC and minimally differs, in the sense of Definition 5.2, from a minimal legal global database instance. If $\mathcal{G}$ is already consistent, then the repairs are the elements of $Mininst(\mathcal{G})$. In Definition 6.6, we are not requiring that a repair respects the property of the sources of being open, i.e., that the extension of each view in the repair contains the corresponding view extension in the source. Thus, it may be the case that a repair – still a global instance – does not belong to $Linst(\mathcal{G})$. If we do not allow this flexibility, a global system might not be repairable. Repairs are used as an auxiliary concept to define the notion of consistent answer.

Example 6.17 (example 6.1 continued) The only element in $Mininst(\mathcal{G}_1)$ is $D_0 = \{R(a,b), R(c,d), R(a,c), R(d,e)\}$. The instance D_0 does not satisfy IC. Hence, $\mathcal{G}_1$ is inconsistent. The repairs are the global instances that minimally differ from D_0 and satisfy the FD, namely $D_0^1 = \{R(a,b), R(c,d), R(d,e)\}$ and $D_0^2 = \{R(a,c), R(c,d), R(d,e)\}$. Notice that they do not belong to $Linst(\mathcal{G}_1)$. $\qquad\square$

Definition 6.7 [Bertossi *et al.*, 2002] (a) Given a global system $\mathcal{G}$, a set of global integrity constraints IC, and a global first-order query $Q(\bar{x})$, we say that a (ground) tuple $\bar{t}$ is a *consistent answer* to Q with respect to IC iff for every repair D of $\mathcal{G}$, $D \models Q[\bar{t}]$. (b) We denote by $Consis_{\mathcal{G}}(Q)$ the set of consistent answers to Q in $\mathcal{G}$. $\square$

Example 6.18 (example 6.17 continued) For the query $Q_1(x) \colon \exists y\ R(x,y)$, the consistent answers are a, c, d. $Q_2(x,y)$: $R(x,y)$ has $(c,d), (d,e)$ as consistent answers. $\square$

If $\mathcal{G}$ is consistent with respect to IC, then $Consis_{\mathcal{G}}(Q) = Minimal_{\mathcal{G}}(Q)$. Furthermore, if the ICs IC are generic, i.e., there is no literal L such that $IC \models L$, then for any $\mathcal{G}$ it holds that $Consis_{\mathcal{G}}(Q) \subseteq Minimal_{\mathcal{G}}(Q)$ [Bertossi *et al.*, 2002]. Notice also that the notion of consistent answer can be applied to queries expressed in Datalog or its extensions with built-ins and negation.

6.2 Specification of Minimal Instances

The specification of the class $Mininst(\mathcal{G})$ for system $\mathcal{G}$ is given using normal logic programs, whose rules are inspired by the inverse-rules algorithm. They use auxiliary predicates instead of function symbols, but their functionality is enforced using the choice predicate [Giannotti *et al.*, 1991]. We consider global system all of whose sources are open.

6.2.1 The Simple Program

In this section, we will present a first approach to the specification of legal instances. In Section 6.2.2, we present the definitive program, that refines the one given in this section. We proceed in this way, because the program we give now, although it may not be suitable for all situations (as discussed later in this section), is simpler to understand than its refined version, already contains the key ideas, and can correctly be applied in some situations which we will characterize.

Definition 6.8 Given an open global system $\mathcal{G}$, the logic program $\Pi(\mathcal{G})$, contains the following facts and clauses:

1. Fact $dom(a)$, for every constant $a \in \mathcal{U}$.[2] Also fact $V_i(\bar{a})$ whenever $\bar{a} \in v_i$ for some source extension v_i in $\mathcal{G}$.

---

[2]Note that, in a DIS, $\mathcal{U}$ does not contain *null*

2. For every view (source) predicate V_i in the system with description $V_i(\bar{x}) \leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$, the rules

$$P_j(\bar{x}_j) \leftarrow V_i(\bar{x}), \ \bigwedge_{z_l \in (\bar{x}_j \setminus \bar{x})} F_i^l(\bar{x}, z_l), \quad j = 1, \ldots n.$$

3. For every predicate $F_i^l(\bar{x}, z_l)$ introduced in 2., the rule

$$F_i^l(\bar{x}, z_l) \leftarrow V_i(\bar{x}), dom(z_l), choice((\bar{x}), (z_l)). \qquad \square$$

In this specification, the predicate $F_i^l(\bar{x}, z_l)$ replaces the Skolem function based atom $f_i^l(\bar{x}) = z_l$ introduced in Section 6.1.4, and, via the choice predicate, it assigns values in the domain to the variables in the head of the rule in 3. that are not in $\bar{x}$. There is a new Skolem predicate for each pair formed by a description rule as in item 2. above and a different existentially quantified variable in it. The predicate $choice((\bar{x}), (z_l))$ ensures that for every (tuple of) value(s) for $\bar{x}$, only one (tuple of) value(s) for z_l is non-deterministically chosen from the constants in the active domain.

Example 6.19 (examples 6.12 and 6.13 continued) Program $\Pi(\mathcal{G}_2)$ contains the following rules:

1. $dom(a). \ dom(b). \ dom(c). \ dom(u). \quad V_1(a, b). \ V_2(a, c).$

2. $P(x, z) \leftarrow V_1(x, y), F_1(x, y, z).$
   $R(z, y) \leftarrow V_1(x, y), F_1(x, y, z).$
   $P(x, y) \leftarrow V_2(x, y).$

3. $F_1(x, y, z) \leftarrow V_1(x, y), dom(z), \ choice((x, y), (z)).$

In this section, we will restrict ourselves to a finite domain $\mathcal{U}$, which is necessary to run the program in real implementations. In this example, we have $\mathcal{U} = \{a, b, c, u\}$ (the extension of predicate dom). In Section 6.5.2, we study how to handle infinite domains by adding to the active domain a finite number of extra constants, like the constant u here. $\qquad \square$

For every program Π with the choice operator, there is a *stable version* $SV(\Pi)$, whose stable models correspond to the so-called *choice models* of Π [Giannotti *et al.*, 1991]. The program $SV(\Pi)$ is obtained as follows:

1. Each choice rule $r:$ $H \leftarrow B, choice((\bar{x}),(y))$ in Π is replaced by the rule
   $H \leftarrow B, chosen_r(\bar{x}, y)$.

2. For each rule as in (a), the following rules are added

   $chosen_r(\bar{x}, y) \leftarrow B, not\ diffChoice_r(\bar{x}, y)$.

   $diffChoice_r(\bar{x}, y) \leftarrow chosen_r(\bar{x}, y'), y \neq y'$.

The rules defined in (2.) ensure that, for every tuple $\bar{x}$ where B is satisfied, the predicate $chosen_r(\bar{x}, y)$ satisfies the functional dependency $\bar{x} \rightarrow y$.

Example 6.20 (example 6.19 continued) Program $SV(\Pi(\mathcal{G}_2))$ contains the following rules:

1. $dom(a)$. $dom(b)$. $dom(c)$. $dom(u)$. $V_1(a, b)$. $V_2(a, c)$.

2. $P(x, z) \leftarrow V_1(x, y), F_1(x, y, z)$.
   $R(z, y) \leftarrow V_1(x, y), F_1(x, y, z)$.
   $P(x, y) \leftarrow V_2(x, y)$.

3. $F_1(x, y, z) \leftarrow V_1(x, y), dom(z), chosen_1(x, y, z)$.

4. $chosen_1(x, y, z) \leftarrow V_1(x, y), dom(z), not\ diffChoice_1(x, y, z)$.
   $diffChoice_1(x, y, z) \leftarrow chosen_1(x, y, z'), dom(z), z' \neq z$.

Its stable models are:

$$\mathcal{M}_1 = \{dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a,b),\ V_2(a,c),\ \underline{P(a,c)},$$
$$diffChoice_1(a,b,a),\quad chosen_1(a,b,b),\quad diffChoice_1(a,b,c),$$
$$diffChoice_1(a,b,u),\ F_1(a,b,b),\ \underline{R(b,b)},\ \underline{P(a,b)}\},$$

$$\mathcal{M}_2 = \{dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a,b),\ V_2(a,c),\ \underline{P(a,c)},$$
$$chosen_1(a,b,a),\quad diffChoice_1(a,b,b),\quad diffChoice_1(a,b,c),$$
$$diffChoice_1(a,b,u),\ F_1(a,b,a),\ \underline{R(a,b)},\ \underline{P(a,a)}\},$$

$$\mathcal{M}_3 = \{dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a,b),\ V_2(a,c),\ \underline{P(a,c)},$$
$$diffChoice_1(a,b,a),\quad diffChoice_1(a,b,b),\quad chosen_1(a,b,c),$$
$$diffChoice_1(a,b,u),\ F_1(a,b,c),\ \underline{R(c,b)}\},$$

$$\mathcal{M}_4 = \{dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a,b),\ V_2(a,c),\ \underline{P(a,c)},$$
$$diffChoice_1(a,b,a),\quad diffChoice_1(a,b,b),\quad diffChoice_1(a,b,c),$$
$$chosen_1(a,b,u),\ F_1(a,b,u),\ \underline{R(u,b)},\ \underline{P(a,u)}\}.$$

The underlined atoms of the models correspond to the elements in which we are interested, namely the global relations of the integration system. □

Definition 6.9 The global instance associated to a choice model $\mathcal{M}$ of $\Pi(\mathcal{G})$ is
$$D_{\mathcal{M}} = \{P(\bar{a}) \mid P \in \mathcal{R} \text{ and } P(\bar{a}) \in \mathcal{M}\}. \qquad \square$$

Example 6.21 (example 6.20 continued) $D_{\mathcal{M}_1}, D_{\mathcal{M}_2}, D_{\mathcal{M}_3}, D_{\mathcal{M}_4}$ are the elements of $Mininst(\mathcal{G}_3)$, namely $\{P(a,b),\ R(b,b),\ P(a,c)\}, \{P(a,a),\ R(a,b),\ P(a,c)\}, \{P(a,c),\ R(c,b)\}, \{P(a,u),\ R(u,b),\ P(a,c)\}$, respectively. □

Theorem 6.1 It holds that
$$Mininst(\mathcal{G}) \subseteq \{D_{\mathcal{M}} \mid \mathcal{M} \text{ is a choice model of } \Pi(\mathcal{G})\} \subseteq Linst(\mathcal{G}).$$

Proof: Consider $\Pi(G)$ as in Definition 6.8. First we prove:

$$\{D_{\mathcal{M}} \mid \mathcal{M} \text{ is a choice model of } \Pi(\mathcal{G})\} \subseteq Linst(\mathcal{G}). \qquad (6.1)$$

Assume that there is a stable model $\mathcal{M}$ of $\Pi(\mathcal{G})$ such that its associated database $D_{\mathcal{M}}$ is not a legal instance. Then there is a view V_i for which $v_i \not\subseteq \varphi_i(D_{\mathcal{M}})$, that is, for some $\bar{a}$:

- $\bar{a} \in v_i$, and then by rules 1. of $\Pi(G)$, $V_i(\bar{a})$ is true in any model of the program, in particular, in $\mathcal{M}$.

- $\bar{a} \notin \varphi_i(D_{\mathcal{M}})$, i.e., in $\mathcal{M}$ it holds that $\neg \exists \bar{z}(P_1(\bar{a}_1, \bar{z}_1) \wedge \ldots \wedge P_n(\bar{a}_n, \bar{z}_n))$, for $\bar{a}_i \subseteq \bar{a}$, and $\bar{z}_i \subseteq \bar{z}$. This is equivalent to

$$\forall \bar{z}(\neg P_1(\bar{a}_1, \bar{z}_1) \vee \ldots \vee \neg P_n(\bar{a}_n, \bar{z}_n)) \text{ being true in } \mathcal{M} \tag{6.2}$$

As a consequence of (6.2) and rules 2. of $\Pi(\mathcal{G})$, the following holds in $\mathcal{M}$:

$$\forall \bar{z} \; (\neg V_i(\bar{a}) \vee \bigvee_l \neg F_i^l(\bar{a}, z_l)). \tag{6.3}$$

Since $V_i(\bar{a}) \in \mathcal{M}$ and rules 3. of $\Pi(G)$ are satisfied by $\mathcal{M}$, for some b's in the domain the atoms $F_i^l(\bar{a}, b) \in \mathcal{M}$. But we had that equation (6.3) holds. We have reached a contradiction because (6.3) is false in $\mathcal{M}$; and (6.1) is proven.

Now we want to prove: $Mininst(\mathcal{G}) \subseteq \{D_{\mathcal{M}} \mid \mathcal{M} \text{ is a choice model of } \Pi(\mathcal{G})\}$.

The program $\Pi(\mathcal{G})$ can be split [Lifschitz and Turner, 1994] into the bottom program Π_B, that contains the facts and rules in 1. and 3. of $\Pi(G)$, and the top program, Π_T, that contains the rules in 2.. If $\mathcal{M}_B$ is a stable model of Π_B and $\mathcal{M}_T^B$ is a stable model of $\Pi_T^{\mathcal{M}_B}$ (the top program partially evaluated by the atoms in $\mathcal{M}_B$), then $\mathcal{M}_B \cup \mathcal{M}_T^B$ is a stable model of $\Pi(\mathcal{G})$, and all the models of latter can be obtained in this way. The bottom program contains the choice operator, and therefore its stable models will correspond to all the possible combinations of values for the Skolem predicates subject to the condition of functionality [Wang and Zaniolo, 2000].

Since $\Pi_T^{\mathcal{M}_B}$ is a non-disjunctive-positive program (without the choice operator), there will be a unique stable model for each $\mathcal{M}_B$ that will correspond to its minimal model.

We will now prove that every minimal legal instance is of the form $D_{\mathcal{M}}$, where $\mathcal{M}$ is of the form $\mathcal{M}_B \cup \mathcal{M}_T^B$, with $\mathcal{M}_B$ a stable model of Π_B and $\mathcal{M}_T^B$ a minimal model of $\Pi_T^{\mathcal{M}_B}$.

Let D be a minimal legal instance of $\mathcal{G}$. Let us define a structure $\mathcal{M}$ for the program $\Pi(\mathcal{G})$ containing the following ground atoms:

1. The atoms in D;

2. $V_i(\bar{a})$ whenever $\bar{a} \in v_i$, where v_i is a source extension in $\mathcal{G}$;

3. $dom(a)$ for every constant $a \in \mathcal{U}$;

4. For each view $V_i(\bar{x})$, consider the rules $F_i^l(\bar{x}, z_l) \leftarrow body(\varphi_{V_i})$, for each variable z_l from the body that does not belong to $\bar{x}$. Evaluate the bodies according to the atoms in 1. When the body is true, add to $\mathcal{M}$ the corresponding atom in the head.

5. If for a view V_i, $\bar{a} \in v_i$ and $F_i^l(\bar{a}, b) \in \mathcal{M}$, add $choice(\bar{a}, b)$ to $\mathcal{M}$.

Note that $D_{\mathcal{M}} = D$. Now we have to prove that the structure $\mathcal{M}$ is a stable model of $\Pi(\mathcal{G})$. This can be shown by proving, first, that $\mathcal{M}_B := (\mathcal{M} \smallsetminus D)$ is a stable model of Π_B, and, next, that $\mathcal{M}_T^{\mathcal{M}_B} = D$ is a minimal model of $\Pi_T^{\mathcal{M}_B}$.

Π_B contains rules 1. and 3. of $\Pi(\mathcal{G})$. By construction, $\mathcal{M}_B$ will satisfies rules 1. For $\mathcal{M}_B$ to satisfy rules 3, it is sufficient to prove that for each $V_i(\bar{a}) \in \mathcal{M}_B$ there is exactly one $F_i^l(\bar{a}, b) \in \mathcal{M}_B$ with $b \in \mathcal{U}$ and that, if $V_i(\bar{a}) \notin \mathcal{M}$, then there is no $F_i^l(\bar{a}, z)$ in $\mathcal{M}_B$. This is enough because it is proven that the choice operator will enforce that $F_i^l(\bar{x}, z)$ satisfies a functional dependency between $\bar{x}$ and z.

Let us suppose, by contradiction, that for $V_i(\bar{a}) \in \mathcal{M}_B$ there are two atoms $F_i^l(\bar{a}, b_1) \in \mathcal{M}_B$ and $F_i^l(\bar{a}, b_2) \in \mathcal{M}_B$. This would imply by construction of $\mathcal{M}$ that the following rules are satisfied by evaluating the bodies with the elements of D: $F_i^l(\bar{a}, b_l) \leftarrow body(\varphi_{V_i})$ and $F_i^l(\bar{a}, b_2) \leftarrow body(\varphi_{V_i})$. This would imply that D has two set of atoms satisfying the mapping $V_i(\bar{a}) \leftarrow body(\varphi_{V_i})$, and therefore D is not minimal. Since D is minimal we have reached a contradiction.

Now we have to prove that if $V_i(\bar{a}) \notin \mathcal{M}$, then there is no $F_i^l(\bar{a}, z)$ in $\mathcal{M}_B$. Let us suppose by contradiction that there for a given value $b \in \mathcal{U}$, $F_i^l(\bar{a}, b) \in \mathcal{M}_B$. This would imply by construction of $\mathcal{M}$ that it holds, by evaluating the bodies with the elements of D, $F_i^l(\bar{a}, b) \leftarrow body(\varphi_{V_i})$. This implies that D satisfies $body(\varphi_{V_i})$ without $V_i(a)$ belonging to the source. Then D is not minimal. Since D is minimal we have reached a contradiction. This proves that $\mathcal{M}_B := (\mathcal{M} \smallsetminus D)$ is a stable model of Π_B. Now we have to prove that D is a minimal model of $\Pi_T^{\mathcal{M}_B}$.

The program $\Pi_T^{\mathcal{M}_B}$ contains only facts of the form $V_i(\bar{a}, \bar{b}) \leftarrow$, where $V_i(\bar{a}) \in \mathcal{M}_B$ and $\bar{b}$ is constructed from all the function predicates $F_i^l(\bar{a}, b_1) \in \mathcal{M}_B$. By construction, this facts are exactly the elements of D. Then, D is a minimal model of $\Pi_T^{\mathcal{M}_B}$. This proves that $\mathcal{M}$ is a stable model of $\Pi(\mathcal{G})$ and since $D_{\mathcal{M}} = D$ every minimal legal instance has a stable model of $\Pi(\mathcal{G})$ associated. $\square$

From the inclusion in Theorem 6.1 it is clear that for monotone queries Q, answers obtained using $\Pi(\mathcal{G})$ under the skeptical or cautious stable model semantics -that sanctions as true what is true of all the stable models of the program- coincide with $Certain_{\mathcal{G}}(Q)$ and $Minimal_{\mathcal{G}}(Q)$. This may not be the case for queries with negation, as pointed out in the remark after Definition 6.4.

In Example 6.21, the stable models are in a one-to-one correspondence with the minimal legal instances, but this may not be always the case.

Example 6.22 Consider an integration system $\mathcal{G}_3$ with global schema $\mathcal{R} = \{P\}$. The set $\mathcal{V}$ of local view definitions consists of $V_1(x) \leftarrow P(x, y)$, and $V_2(x, y) \leftarrow P(x, y)$, with source contents $v_1 = \{V_1(a)\}$, $v_2 = \{V_2(a, c)\}$, resp. We have that $Mininst(\mathcal{G}_3) = \{\{P(a, c)\}\}$. However, the global instances corresponding to models of $\Pi(\mathcal{G}_3)$ are of the form $\{\{P(a, c), P(a, z)\} \mid z \in \mathcal{U}\}$. As V_2 is open, it forces $P(a, c)$ to be in all legal instances, and with this, the same condition on V_1 is automatically satisfied, and no other values for y are needed. But the choice operator still has freedom to chose other values (the $z \in \mathcal{U}$). This is why we get more legal instances than the minimal ones. □

Now we investigate sufficient conditions under which the simple program of Definition 6.8 captures the minimal instances. This is important because the general program to be presented in Section 6.2.2 is much more complex than the simple version presented so far.

Definition 6.10 We define a *section of a view* V_i as a set S_i^l consisting either of all the predicates in the body of its definition that share the same existential variable z_l or all the atoms without existential variables, in which case $l = 0$ and the view section is denoted with S_i^0. □

For example, the view defined by $V(x, y) \leftarrow P(x, z_1), R(z_1, y), T(x, y)$ has two sections: $S_1^1 = \{P(x, z_1), R(z_1, y)\}$ and $S_1^0 = \{T(x, y)\}$. Let Sec denote the set of all view sections for system $\mathcal{G}$.

Given a view section S_i^l, we denote by $Const(S_i^l)$, $UVar(S_i^l)$ and $EVar(S_i^l)$ the sets of constants, universal variables, and existential variables, respectively, that occur in predicates in S_i^l.

Let μ, ε be two new constants. For a view section S_i^l, an *admissible mapping* is any mapping $h\colon Const(S_i^l) \cup UVar(S_i^l) \cup EVar(S_i^l) \rightarrow Const(S_i^l) \cup \{\mu, \varepsilon\}$, such that: (a)

$h(c) = c$, for every $c \in Const(S_i^l)$; (b) $h(x) = D$ with $D \in Const(S_i^l) \cup \{\mu\}$, for every $x \in UVar(S_i^l)$; (c) $h(z) = F$ with $F \in Const(S_i^l) \cup \{\mu, \varepsilon\}$, for every $z \in EVar(S_i^l)$.

A particular admissible mapping L is given by (a) $L(c) = c$, for every $c \in Const(S_i^l)$; (b) $L(x) = \mu$, for every $x \in UVar(S_i^l)$; (c) $L(z) = \varepsilon$, for every $z \in EVar(S_i^l)$. For an admissible mapping h, $h(S_i^l)$ denotes the set of atoms obtained from S_i^l by applying h to the arguments in S_i^l.

Theorem 6.2 Given an integration system $\mathcal{G}$, if for every view section S_i^l with existential variables there is no admissible mapping h for S_i^l, such that

$$h(S_i^l) \subseteq \bigcup_{S \in (Sec \smallsetminus \{S_i^l\})} L(S),$$

then the instances associated to the stable models of the simple version of $\Pi(\mathcal{G})$ are exactly the minimal legal instances of $\mathcal{G}$.

Proof: Let us suppose by contradiction that we have an integration system G that has no admissible mapping h for S_i^l (with $i \neq 0$), such that $h(S_i^l) \subseteq \bigcup_{S \in (Sec \smallsetminus \{S_i^l\})} L(S)$, and that there is a stable model $\mathcal{M}$ of the simple version of $\Pi(\mathcal{G})$ such that the database associated $D_\mathcal{M}$ is not a minimal legal instance.

Since $D_\mathcal{M}$ is not minimal, there is a minimal legal instance E such that $E \subsetneq D_\mathcal{M}$. It follows from Theorem 6.1 that there is a model $\mathcal{M}'$ of $\Pi(\mathcal{G})$ such that $D_{\mathcal{M}'} = E$. Then, there should be a non empty set $\mathcal{C}$, such that $\mathcal{C} \in \mathcal{M}$ and $\mathcal{C} \notin \mathcal{M}'$.

It follows from the proof of Theorem 6.1 that the program $\Pi(\mathcal{G})$ can be divided into two parts Π_B and $\Pi_T^{\mathcal{M}_B}$, where the second is a result of an evaluation of the model $\mathcal{M}_B$ of Π_B over the rules of $\Pi(\mathcal{G})$ that do not belong to Π_B. The interesting thing is that the program $\Pi_T^{\mathcal{M}_B}$ turns out to be a set of facts of global relations. This shows that the different models will be determined only by the functional predicates atoms of the form $F_i^l(\bar{a}, b)$ chosen in each model. Each of this atom will generate exactly one

global atom for each relation that has the existential variable z_l in the view V_i. Then, the only way that one model might generate a legal instance of $\mathcal{G}$ with less elements than other model is if two functional predicate atoms generate the same global atom. Then, $\mathcal{C}$ has to be formed by instantiations of sections with existential variables. For simplicity and without lost of generality, let us suppose that $\mathcal{C}$ has exactly one instantiation of one section. For $\mathcal{C}$ to belong to $\mathcal{M}$ and not to $\mathcal{M}'$, $\mathcal{M}$ should have different values of the existential variables that generate the instantiations of $\mathcal{C}$ than those assigned in $\mathcal{M}$, and the rest values should be the same (since $D_{\mathcal{M}'} \subsetneq D_{\mathcal{M}}$). Furthermore, the values given in $\mathcal{M}'$ should generate the same set of predicates that another section or sections generates in $\mathcal{M}$ and in $\mathcal{M}'$. Then, if $\mathcal{C}$ is the instantiation of a section S_i^l, the following has to hold for every value a_k in position k of the atom $P(\bar{a}) \in \mathcal{C}$, being this atom an instantiation of $P(x_1, \ldots, x_k, \ldots, x_n) \in S_i^l$:

1. If $x_k \in Const(S_i^l)$, then there is a different section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$ and $x_k \in Const(S_j^m)$ and $x_k = a_k$.

2. If $x_k \in UVar(S_i^l)$, then there are two options:

   (a) There is other section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$, $x_k \in Const(S_j^m)$ and $x_k = a_k$.

   (b) There is other section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$, $x_k \in UVar(S_j^m)$ and $(\ldots, a_k, \ldots) \in v_j$.

3. If $x_k \in EVar(S_i^l)$, then there are three options:

   (a) There is other section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$, $x_k \in Const(S_j^m)$ and $x_k = a_k$.

   (b) There is other section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$, $x_k \in UVar(S_j^m)$ and $(\ldots, a_k, \ldots) \in v_j$.

(c) There is other section S_j^m such that $P(\ldots, x_k, \ldots) \in S_j^m$, $x_k \in EVar(S_j^m)$ and $F_j^k(\bar{b}, a_k) \in \mathcal{M}'$ for $(\bar{b}) \in v_j$.

Consider a mapping h defined by the different cases just described. For example, if we are in case (2b), we have that $h(x_k) = \mu$, and in case (3a), $h(x_k) = a_k$. By construction, this mapping is such that $h(S_i^l) \subseteq \bigcup_{S \in (Sec \smallsetminus \{S_i^l\})} L(S)$. We have reached a contradiction since we assumed the mapping h did not exist. $\qquad \square$

Basically, Theorem 6.2 says that if there is an admissible mapping with $h(S_i^l) \subseteq \bigcup_{S \in (Sec \smallsetminus \{S_i^l\})} L(S)$, then it is possible to have some view contents for which the openness will be satisfied by the other sections in Sec, and then it will not be necessary to compute values for the existential variables in section S_i^l. Since the simple version will always compute values for them, it may specify more legal instances than the minimal ones.

Example 6.23 (example 6.22 continued) The first view is defined by $V_1(x) \leftarrow P(x, y)$, and has only one section $S_1^y = \{P(x, y)\}$. For the admissible mapping h defined by $h(x) = h(y) = \mu$, we have $h(S_1^y) = \{P(\mu, \mu)\} \subseteq L(S_2^0)$. The conditions of the theorem are not satisfied, and there is no guarantee that the simple version will calculate exactly the minimal instances of $\mathcal{G}_3$. Actually, we already know that this is not the case. $\qquad \square$

Example 6.24 (examples 6.12 and 6.13 continued) There are two view sections: $S_1^z = \{P(x, z), Q(z, y)\}$ and $S_2^0 = \{P(x, y)\}$, where x and y are universal variables and z is an existential variable. It is easy to see that there is no mapping h for which $h(S_1^z) \subseteq L(S_2^0)$ nor $h(S_2^0) \subseteq L(S_1^z)$. As a consequence, for any source contents, the simple version of $\Pi(\mathcal{G}_2)$ will calculate exactly the minimal instances of $\mathcal{G}_2$. $\qquad \square$

6.2.2 The Refined Program

In the general case, if we want to compute only the elements of $Mininst(\mathcal{G})$, we need to refine the program $\Pi(\mathcal{G})$ given in the previous section. For this purpose, we will introduce auxiliary annotation constants that will be used as extra arguments in the database predicates. The annotation constants and their meaning are described in Table 6.1.

annotation	atom	the tuple $P(\bar{a})$ is ...
$\mathbf{t_o}$	$P\_(\bar{a}, \mathbf{t_o})$	is an obligatory atom in all the minimal legal instances
$\mathbf{v_i}$	$P\_(\bar{a}, \mathbf{v_i})$	an optional atom introduced to satisfy the openness of view v_i
$\mathbf{nv_i}$	$P\_(\bar{a}, \mathbf{nv_i})$	an optional atom introduced to satisfy the openness of view that is not v_i

Table 6.1: Annotation constants and their meanings for DIS

Definition 6.11 Given an open global system $\mathcal{G}$, the refined program $\Pi(\mathcal{G})$, contains the following clauses:

1. Fact $dom(a)$, for every constant $a \in \mathcal{U}$.[3]

2. Fact $V_i(\bar{a})$, whenever $\bar{a} \in v_i$ for some source extension v_i in $\mathcal{G}$.

3. For every view (source) predicate V_i in the system with description $V_i(\bar{x}) \leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$:

    (a) For every P_k with no existential variables, the rules

    $$P_{k\_}(\bar{x}_k, \mathbf{t_o}) \leftarrow V_i(\bar{x}).$$

    (b) For every set S_{ij} of predicates of the description's body that are related by common existential variables $\{z_1, \ldots, z_m\}$, the rules,

---

[3]Note that, in a DIS, $\mathcal{U}$ does not contain *null*

$$P_{k\_}(\bar{x}_k, \mathbf{v_{ij}}) \leftarrow add_{v_{ij}}(\bar{x}'), \; \bigwedge_{z_l \in (\bar{x}_k \setminus \bar{x}')} F_i^l(\bar{x}', z_l), \text{ for } P_k \in S_{ij}.$$

$$add_{v_{ij}}(\bar{x}') \leftarrow V_i(\bar{x}), \; not \; aux_{v_{ij}}(\bar{x}').$$

$$aux_{v_{ij}}(\bar{x}') \leftarrow \bigwedge_{l=1}^{m} var_{v_{ij}z_l}(\bar{x}_{z_l}).$$

$$var_{v_{ij}z_l}(\bar{x}_{z_l}) \leftarrow \bigwedge_{P_k \in S_{ij}\&z_l \in \bar{x}_k} P_{k\_}(\bar{x}_k, \mathbf{nv_{ij}}).$$

where $\bar{x}_{z_l} = \bigcup_{P_k \in S_{ij}\&z_l \in \bar{x}_k} \bar{x}_k$, for $l = 1, \ldots m$ and $\bar{x}' = \bar{x} \cap \bigcup_{P_k \in S_{ij}} \bar{x}_k$.

4. For every predicate $F_i^l(\bar{x}', z_l)$ introduced in 3.(b), the rules

$$F_i^l(\bar{x}', z_l) \leftarrow add_{v_{ij}z_l}(\bar{x}'), dom(z_l), choice((\bar{x}'), (z_l)).$$

$$add_{v_{ij}z_l}(\bar{x}') \leftarrow add_{v_{ij}}(\bar{x}'), \; not \; aux_{v_{ij}z_l}(\bar{x}'), \text{ for } l = 1, \ldots m.$$

$$aux_{v_{ij}z_l}(\bar{x}') \leftarrow var_{v_{ij}z_l}(\bar{x}_{z_l}), \bigwedge_{z_k \neq z_l \& z_k \in \bar{x}_{z_l}} F_i^k(\bar{x}', z_k), \text{ for } l = 1, \ldots m.$$

5. For every global relation $P(\bar{x})$, the rules

$$P_\_(\bar{x}, \mathbf{nv_{ij}}) \leftarrow P_\_(\bar{x}, \mathbf{v_{hk}}), \quad \text{for } \{(ij, hk) \mid P(\bar{x}) \in S_{ij} \cap S_{hk}, ij \neq hk\}.$$

$$P_\_(\bar{x}, \mathbf{nv_{ij}}) \leftarrow P_\_(\bar{x}, \mathbf{t_o}), \quad \text{for } \{(ij) \mid P(\bar{x}) \in S_{ij}\}.$$

$$P(\bar{x}) \leftarrow P_\_(\bar{x}, \mathbf{v_{ij}}), \quad \text{for } \{(ij) \mid P(\bar{x}) \in S_{ij}\}.$$

$$P(\bar{x}) \leftarrow P_\_(\bar{x}, \mathbf{t_o}). \hspace{4cm} \Box$$

Example 6.25 (example 6.22 continued) The refined program $\Pi(\mathcal{G}_3)$ is:

1. $dom(a). \qquad dom(c).$

2. $v_1(a). \qquad v_2(a, c).$

3. $P_\_(x, z, \mathbf{v_1}) \leftarrow add_{v_1}(x), F_z(x, z).$

   $add_{v_1}(x) \leftarrow v_1(x), \; not \; aux_{v_1}(x).$

   $aux_{v_1}(x) \leftarrow var_{v_1z}(x, z).$

   $var_{v_1z}(x, z) \leftarrow P(x, z, nv_1).$

4. $F_z(x, z) \leftarrow add_{v_1}(x), dom(z), chosen_{v_1z}(x, z).$

   $chosen_{v_1z}(x, z) \leftarrow add_{v_1}(x), dom(z), \ not \ diffchoice_{v_1z}(x, z).$

   $diffchoice_{v_1z}(x, z) \leftarrow chosen_{v_1z}(x, z'), dom(z), z' \neq z.$

5. $P_{\_}(x, y, \mathbf{t_o}) \leftarrow v_2(x, y).$

   $P_{\_}(x, y, \mathbf{nv_1}) \leftarrow P_{\_}(x, y, \mathbf{t_o}).$

   $P(x, y) \leftarrow P_{\_}(x, y, \mathbf{v_1}).$

   $P(x, y) \leftarrow P_{\_}(x, y, \mathbf{t_o}).$

Rules in 3. ensure that if there is an atom in source V_1, e.g., $V_1(\bar{a})$, and if an atom of the form $P(\bar{a}, z)$ was not added by view V_2, then it is added by the first rule in 3. with a z value given by the function predicate $F_z(\bar{a}, z)$. This function predicate is calculated by rules in 4. The first rule in 5. enforces the satisfaction of the openness of V_2, by adding obligatory atoms to predicate P; and rule 2. in 5. stores this atoms with the annotation $\mathbf{nv_1}$, implying that they were added by a view different from V_1. The last two rules gather in the global relations the elements that were generated by both views and that are in the minimal legal instances. The stable model of this program is $\{dom(a), dom(c), v_1(a), v_2(a, c), \underline{P(a, c)}, P_{\_}(a, c, \mathbf{t_o}), P_{\_}(a, c, \mathbf{nv_1}), aux_{v_1}(a)\}$, which corresponds to the only minimal legal instance $\{P(a, c)\}$. □

Theorem 6.3 If $\mathcal{M}$ is a stable model of $SV(\Pi(\mathcal{G}))$, then $D_{\mathcal{M}} := \{P(\bar{a}) \mid P \in \mathcal{R}$ and $P(\bar{a}) \in \mathcal{M}\} \in Mininst(\mathcal{G})$. Furthermore, the minimal legal instances obtained in this way are all the minimal legal instances of $\mathcal{G}$. □

The following lemmas are needed in order to prove Theorem 6.3.

Lemma 6.1 If $\mathcal{M}$ is a stable model of $SV(\Pi(\mathcal{G}))$, then $D_{\mathcal{M}}$ is a legal instance of $\mathcal{G}$.

Proof: In the proof we use the same notation as in the Definition 6.11 of $\Pi(\mathcal{G})$. Assume that $D_{\mathcal{M}}$ is not legal. Then there must be a view V_i, with definition $\varphi_i \colon V_i(\bar{x}) \leftarrow$

$\bigwedge_{u=1}^{n} P_u(\bar{x}_u, \bar{z}_u)$, for which $v_i \not\subseteq \varphi_i(D_{\mathcal{M}})$. More specifically, there is $\bar{a}$ such that $\bar{a} \in (v_i \smallsetminus \varphi_i(D_{\mathcal{M}}))$. If $\bar{a} \in v_i$ then $V_i(\bar{a}) \in \mathcal{M}$.

For every global relation P_u without existential variables in the view definition φ_i, we can conclude from rules 3(a) of $\Pi(\mathcal{G})$ that $P_{u\_}(\bar{a}_u, \mathbf{t_o}) \in \mathcal{M}$ with $\bar{a}_u \subseteq \bar{a}$. Then, by rules 5., $P_u(\bar{a}_u) \in \mathcal{M}$ and therefore $P_u(\bar{a}_u) \in D_{\mathcal{M}}$.

Now we will analyze the case of global relation with existential variables treated by rules defined in 3(b). For a certain S_{ij}, in order to satisfy the second rule of 3(b), we have to analyze two cases:

1. $V_i(\bar{a}) \in \mathcal{M}$ and $aux_{v_{ij}}(\bar{a}') \notin \mathcal{M}$. Then, from the second rule in 3(b)., $add_{v_{ij}}(\bar{a}') \in \mathcal{M}$. It follows from the third rule of 3(b) that there exists a non-empty set $\mathcal{L}$ such that $var_{v_{ij}z_l}(\bar{a}_{z_l}) \notin \mathcal{M}$ for $l \in \mathcal{L}$. Now let us take a look at rules in 4. From the 3rd rule, we have that for every $l \in \mathcal{L}$, $aux_{v_{ij}z_l}(\bar{a}') \notin \mathcal{M}$. Then, it follows from the 2nd rule and the fact that $add_{v_{ij}}(\bar{a}') \in \mathcal{M}$ that for every $l \in \mathcal{L}$, $add_{v_{ij}z_l}(\bar{a}') \in \mathcal{M}$. Now, from the first rule, the choice operator will assign one value of the domain to z_l, e.g. b_l for each $l \in \mathcal{L}$. Then we will have $F_i^l(\bar{a}', b_l) \in \mathcal{M}$ for every $l \in \mathcal{L}$. Now let us have a look at the rules in 3(b). For $P_k \in S_{ij}$, there are two cases to analyze with respect to the first rule:

   (a) $\{z_l \mid z_l \in (\bar{x}_k \smallsetminus \bar{x}')\} \subseteq \{z_l \mid l \in \mathcal{L}\}$. Then $P_{k\_}(\bar{a}_k, \mathbf{v_{ij}}) \in \mathcal{M}$, where $\bar{a}_k$ is a projection of $\bar{a}$ and the b_l of the functional predicates. Hence, $P_k(\bar{a}_k) \in \mathcal{M}$, and therefore $P_k(\bar{a}_k) \in D_{\mathcal{M}}$.

   (b) $\{z_l \mid z_l \in (\bar{x}_k \smallsetminus \bar{x}')\} \not\subseteq \{z_l \mid l \in \mathcal{L}\}$. For every $z_{l'} \in \{z_l \mid z_l \in (\bar{x}_k \smallsetminus \bar{x}')\} \smallsetminus \{z_l \mid l \in \mathcal{L}\}$,we have $var_{v_{ij}z_{l'}}(\bar{a}_{z_{l'}}) \in \mathcal{M}$ since $l' \notin \mathcal{L}$. Now, since the only way for an atom to belong to a model is to have a rule with it in the head and the body satisfied, the body of the fourth rule of 3(b) has to be true. This implies that $P_{k\_}(\bar{a}_k, \mathbf{nv_{ij}}) \in \mathcal{M}$. We also have that since

$F_i^{l'}(\bar{a}', b_l) \notin \mathcal{M}$ for any value of b_l, then $add_{v_{ij}z_{l'}}(\bar{a}') \notin \mathcal{M}$ and therefore $aux_{v_{ij}z_{l'}}(\bar{a}') \in \mathcal{M}$. Then, it follows form the third rule of 3(b) that the values associated to the existential variables that are not $z_{l'}$ in $P_{k\_}(\bar{a}_k, \mathbf{nv_{ij}})$ coincide with the values given by the functional predicates of the view. Since $P_{k\_}(\bar{a}_k, \mathbf{nv_{ij}}) \in \mathcal{M}$, we have from rules in 5. that $P_{k\_}(\bar{a}_k, \mathbf{nv_{hk}})$ (with $hk \neq ij$) or $P_{k\_}(\bar{a}_k, \mathbf{t_o})$ belong to $\mathcal{M}$, and therefore that $P_k(\bar{a}_k) \in \mathcal{M}$. Then, $P_k(\bar{a}_k) \in D_{\mathcal{M}}$, sharing the same existential variable that those generated by the previews case considered.

Then $\bar{a}_{S_{ij}} \in \varphi_{iS_{ij}}(D_{\mathcal{M}}).$[4]

2. $V_i(\bar{a}) \in \mathcal{M}$ and $aux_{v_{ij}}(\bar{a}') \in \mathcal{M}$. Then, from the second rule in 3(b)., $add_{v_{ij}}(\bar{a}') \notin \mathcal{M}$. From the 3rd rule of 3(b) $var_{v_{ij}z_l}(\bar{a}_{z_l}) \in \mathcal{M}$ for all z_l. Then, from the fourth rule of 3(b) $P_{k\_}(\bar{a}_k, \mathbf{nv_{ij}}) \in \mathcal{M}$ for all $P_k \in S_{ij}$ such that $z_l \in \bar{x}_k$. From rules in 5., with $hk \neq ij$, $P_{k\_}(\bar{a}_k, \mathbf{nv_{hk}})$ or $P_{k\_}(\bar{a}_k, \mathbf{t_o})$ belong to $\mathcal{M}$, and therefore that $P_k(\bar{a}_k) \in \mathcal{M}$. Then, $P_k(\bar{a}_k) \in D_{\mathcal{M}}$ which in it turns implies that $\bar{a}_{S_{ij}} \in \varphi_{iS_{ij}}(D_{\mathcal{M}})$

Now, since the different S_{ij} do not share existential variables, $\varphi_i(D_{\mathcal{M}}) = \bowtie_{S_{ij} \in V_i} \varphi_{iS_{ij}}(D_{\mathcal{M}})$. Then, since $\bar{a}_{S_{ij}} \in \varphi_{iS_{ij}}(D_{\mathcal{M}})$, $\bar{a} \in \varphi_i(D_{\mathcal{M}})$. We have reached a contradiction. $\qquad \square$

Lemma 6.2 If D is a minimal instance of $\mathcal{G}$, then there is a stable model $\mathcal{M}$ of $SV(\Pi(\mathcal{G}))$, such that $D_{\mathcal{M}} = D$.

Proof: We need to define a Herbrand structure that will be our candidate to be the stable model $\mathcal{M}$ that generates instance D. For doing this, we use the same notation

---

[4]We denote by $\bar{a}_{S_{ij}}$ the atom $\bar{a}$ restricted to the variables of the view φ_i that belong to S_{ij}, and by $\varphi_{iS_{ij}}$ the view definition φ_i restricted to the predicates in S_{ij} and its variables

as in the Definition 6.11 of $\Pi(\mathcal{G})$. We put the following facts into $\mathcal{M}$:

1. $P_k(\bar{a})$ for every global atom $P_k(\bar{a}) \in D$. No other atom annotated with P_k belongs to $\mathcal{M}$.

2. $dom(a)$ iff $a \in \mathcal{U}$.

3. $V_i(\bar{a})$ iff $\bar{a} \in v_i$ for $v_i \in \mathcal{G}$.

4. $P_{k\_}(\bar{a}_k, \mathbf{t_o})$ iff there is a view $V_i(\bar{x}) \leftarrow P_1(\bar{x}_1), \ldots, P_k(\bar{x}_k), \ldots P_n(\bar{x}_n)$, in which P_k has no existential variables and such that $\bar{a} \in v_i$.

5. For every atom $P_k(\bar{a}_k) \in D$, where $P_{k\_}(\bar{a}_k, \mathbf{t_o}) \notin \mathcal{M}$, we need to check which views had the potential of generating it. After some considerations we will specify at the end of this item what new atoms go into $\mathcal{M}$ and which do not.

   For each view section S_i^l with an existential variable z_l,[5] such that $P_k \in S_i^l$, define the following views:

$$P_{k\_}(\bar{x}'_k, S_i^l) \leftarrow \bigwedge_{P_j(\bar{x}_j) \in S_i^l} P_j(\bar{x}_j) \wedge V_i(\bar{x}),$$

   where S_i^l is considered as an annotation constant in the second argument of head of the view. Let P be the result of instantiating these views over the atoms in D and the source extensions. P contains the possible section that might have generated the presence of each global atom in D. We will define $S^{P_k} = \{S_i^l \mid P_{k\_}(\bar{a}_k, S_i^l) \in P\}$, i.e., S^{P_k} contains al the sections from which $P_k(\bar{a}_k)$ could have been generated. Note that there is only one S_{ij} in $\mathcal{G}$ such that $S_{ij} \supseteq S_i^m$.[6] Then, for each section $S_i^l \in S^{P_k}$ that does not have an admissible mapping [7]

---

[5] The S_i^l are the view sections introduced in Section 6.2.1.
[6] Here, the S_{ij} are those appearing in Definition 6.11.
[7] As defined in section 6.2.1

such that $h(S_i^l) \subseteq \bigcup_{S \in (Sec \smallsetminus \{S_i^l\})} L(S)$, we do the following: $P_{k\_}(\bar{a}_k, \mathbf{v_{ij}}) \in \mathcal{M}$, $add_{v_{ij}}(\bar{a}') \in \mathcal{M}$, $V_i(\bar{a}) \in \mathcal{M}$, $aux_{v_{ij}}(\bar{a}') \notin \mathcal{M}$, $var_{v_{ij}z_l}(\bar{a}_{z_l}) \notin \mathcal{M}$, $aux_{v_{ij}z_l}(\bar{a}') \notin \mathcal{M}$, $add_{v_{ij}z_l}(\bar{a}') \in \mathcal{M}$. For all the rest of the sections of S^{P_k}, e.g. S_i^m, we have that the $var_{v_{in}z_m}(\bar{a}_{z_m}) \in \mathcal{M}$. If for all the sections in a view $var_{v_{in}z_m}(\bar{a}_{z_m}) \in \mathcal{M}$, then $aux_{v_{ij}}(\bar{a}') \in \mathcal{M}$ and $add_{v_{ij}}(\bar{a}') \notin \mathcal{M}$.

6. For every $P_{k\_}(\bar{a}_k, \mathbf{v_{ij}}) \in \mathcal{M}$, we add the fact $P_{k\_}(\bar{a}_k, \mathbf{nv_{km}})$ to $\mathcal{M}$ for every $S_{km} \neq S_{ij}$.

7. For every $add_{v_{ij}z_l}(\bar{a}'), P_{k\_}(\bar{a}_k, \mathbf{v_{ij}}) \in \mathcal{M}$, add $F_i^l(\bar{x}, z_l)$ into $\mathcal{M}$, where z_l is the value of that existential variable in $P_{k\_}(\bar{a}_k, \mathbf{v_{ij}})$.

By construction, $\mathcal{M}$ minimally satisfies rules 1., 2., 3(a), 5. and the first rule of 3(b) in the program $\Pi(\mathcal{G})^{\mathcal{M}}$. If $aux_{v_{ij}}(\bar{a}') \in \mathcal{M}$, $\Pi(\mathcal{G})^{\mathcal{M}}$ does not include the second type of rules of 3(b). If $aux_{v_{ij}}(\bar{a}') \notin \mathcal{M}$, $\Pi(\mathcal{G})^{\mathcal{M}}$ has the rule $add_{v_{ij}}(\bar{x}') \leftarrow V_i(\bar{x})$ corresponding to second type of rules of 3(b). This rule is satisfied by $\mathcal{M}$ because of the facts added to $\mathcal{M}$ in item 5. For the section S_i^l that has no admissible mapping such that $h(S_i^l) \subseteq \bigcup_{S \in (Sec \smallsetminus \{S_i^l\})} L(S)$, it holds that no other views can generate the facts for this section, and therefore that the body of the fourth rules in 3(b) will not be satisfied. Since in that case $var_{v_{ij}z_l}(\bar{a}_{z_l}) \notin \mathcal{M}$, the whole rule is satisfied. For the sections that are not in this case, i.e.,there is an admissible mapping, then the body of the fourth rules in 3(b) will be satisfied and since $var_{v_{ij}z_l}(\bar{a}_{z_l}) \in \mathcal{M}$, the whole rule will be satisfied. If all the sections are in the situation last described, $aux_{v_{ij}}(\bar{a}') \in \mathcal{M}$ and therefore the third rule in 3(b). will be satisfied. Following the same analysis and the fact that the choice operator will choose any value of the domain, it is easy to see that rules in 4. are also minimally satisfied. $\mathcal{M}$ is a minimal model of $\Pi(\mathcal{G})^{\mathcal{M}}$ and therefore there is a stable model of $\Pi(\mathcal{G})$, $\mathcal{M}$, such that $D_{\mathcal{M}}$ corresponds to the minimal legal instance D. $\square$

Lemma 6.3 If $\mathcal{M}$ is a stable model of $SV(\Pi(\mathcal{G}))$, then $D_{\mathcal{M}}$ is a minimal instance of $\mathcal{G}$.

Proof: The legality of $D_{\mathcal{M}}$ was established in Lemma 6.1. Assume, by contradiction that $D_{\mathcal{M}}$ is not a minimal instance of $\mathcal{G}$. Then there must be a minimal instance D such that $D \subsetneq D_{\mathcal{M}}$. It follows from Lemma 6.2 that there is a model $\mathcal{M}'$ such that $D_{\mathcal{M}'} = D$. Then, $D_{\mathcal{M}'} \subsetneq D_{\mathcal{M}}$. In particular, there is an atom of a global relation, say $P_k(\bar{a})$, such that $P_k(\bar{a}) \in \mathcal{M}$ and $P_k(\bar{a}) \notin \mathcal{M}'$. If $P_k(\bar{a}) \in \mathcal{M}$ we have two options:

1. $P_{k\_}(\bar{a}, \mathbf{t_o}) \in \mathcal{M}$. Then there is a view v_i in which P_k has no existential variables. In that case $P_{k\_}(\bar{a}, \mathbf{t_o})$ belongs to all the models and in particular to $\mathcal{M}'$. We have reached a contradiction since $P_k(\bar{a}) \notin \mathcal{M}'$

2. $P_{k\_}(\bar{a}, \mathbf{v_{ij}}) \in \mathcal{M}$. This implies that $add_{v_{ij}}(\bar{a}') \in \mathcal{M}$ and for all $a_l \in (\bar{a} \setminus \bar{a}')$, $F_i^l(\bar{a}', a_l) \in \mathcal{M}$. Hence there is an atom $V_i(\bar{A}) \in \mathcal{M}$ such that the first rule of 3.b. is satisfied. We can also conclude that $var_{v_{ij}z_l}(\bar{a}_{z_l}) \notin \mathcal{M}$. Then there is no other view that satisfies this section S_i^l. This implies that if $\mathcal{M}'$ does not contain $P_k(\bar{a})$ then, in order to satisfy the openness of view v_i it must add a new atom to predicate P_k. But $D'_{\mathcal{M}} \subsetneq D_{\mathcal{M}}$. We have reached a contradiction

As we reached a contradiction in both cases, we have proven that $D_{\mathcal{M}}$ is a minimal legal instance of $\mathcal{G}$. $\qquad \square$

Proof of Theorem 6.3: Directly from Lemma 6.2 and 6.3 $\qquad \square$

Since Theorem 6.3 holds, the program $\Pi(\mathcal{G})$ (or its stable version) can be used to compute $Minimal_{\mathcal{G}}(Q)$, where Q is a query expressed as a, say Datalog^{not} program $\Pi(Q)$. This can be done by running the combined program under the skeptical stable model semantics. The following corollary for monotone queries, e.g. Datalog queries

with comparisons, can be immediately obtained from Theorem 6.3 and the fact that for those queries, $Certain_{\mathcal{G}}(Q) = Minimal_{\mathcal{G}}(Q)$.

Corollary 6.1 The certain answers to monotone queries posed to an open integration system $\mathcal{G}$ can be computed by running, under the skeptical stable model semantics, the query program in combination with the program $\Pi(\mathcal{G})$ that specifies the minimal legal instances of $\mathcal{G}$. $\square$

We know that under the hypothesis of Theorem 6.2, the simple and refined programs compute the same legal database instances, namely the minimal ones. Under the hypothesis of Theorem 6.2, it is possible to provide a syntactic transformation of the refined program into the simple program (see Appendix B).

6.3   Specification of Repairs of a Global System

In Chapter 5, repairs of single relational databases are specified as stable models of disjunctive logic programs. In the case of data integration systems, we can treat each minimal legal instances as a single database. In this way, we can combine the program that specifies the minimal legal instances with the program that repairs databases.

Definition 6.12 Given a schema $\mathcal{R}$ and a set of ICs IC, we will denote by $\Pi(\mathcal{R}, IC)$ the repair program for single databases but without the facts, i.e., rules 2. to 6. in Definition 5.9. $\square$

$\Pi(\mathcal{R}, IC)$ consists of the intentional part of the repair program. By combining it with the minimal legal instance program, we obtain the repair program for an integration system.

Definition 6.13 Given an integration system $\mathcal{G}$ with global schema $\mathcal{R}$ and a set of global ICs IC, the *repair program* is:

$$\Pi(\mathcal{G}, IC) \; := \; \Pi(\mathcal{G}) \; \cup \; \Pi(\mathcal{R}, IC)$$

$\square$

Example 6.26 (example 6.20 continued) We have the integration system $\mathcal{G}_2$ with the local view definitions $V_1(x, z) \leftarrow P(x, y), R(y, z)$, and $V_2(x, y) \leftarrow P(x, y)$, and source contents $v_1 = \{V_1(a, b)\}$ and $v_2 = \{V_2(a, c)\}$, respectively. Consider the global symmetry integrity constraint $sim : \; \forall x \forall y (R(x, y) \rightarrow R(y, x))$ on $\mathcal{G}_2$. The repair program, $\Pi(\mathcal{G}, IC)$, is the union of $\Pi(\mathcal{G})$ and $\Pi(D, IC)$ below:

$\Pi(\mathcal{G})$:

$\quad dom(a). \quad\quad dom(b). \quad\quad dom(c). \quad\quad dom(u).$

$\quad v_1(a, b). \quad\quad v_2(a, c).$

$\quad P\_(x, y, \mathbf{v_1}) \leftarrow add_{v_1}(x, z), F_1^y(x, z, y).$

$\quad R\_(y, z, \mathbf{v_1}) \leftarrow add_{v_1}(x, z), F_1^y(x, z, y).$

$\quad add_{v_1}(x, z) \leftarrow v_1(x, z), \; not \; aux_{v_1}(x, z).$

$\quad aux_{v_1}(x, z) \leftarrow var_{v_1 y}(x, y, z).$

$\quad var_{v_1 y}(x, y, z) \leftarrow P\_(x, y, \mathbf{nv_1}), R\_(y, z, \mathbf{nv_1}).$

$\quad F_1^y(x, z, y) \leftarrow add_{v_1 y}(x, z), dom(y), chosen_{v_1 y}(x, z, y).$

$\quad chosen_{v_1 y}(x, z, y) \leftarrow add_{v_1 y}(x, z), dom(y), \; not \; diffchoice_{v_1}(x, z, y).$

$\quad diffchoice_{v_1}(x, z, y) \leftarrow chosen_{v_1 y}(x, z, y'), dom(y), y' \neq y.$

$\quad add_{v_1 y}(x, z) \leftarrow add_{v_1}(x, z), \; not \; aux_{v_1 y}(x, z).$

$\quad aux_{v_1 y}(x, z) \leftarrow var_{v_1 y}(x, y, z).$

$\quad P\_(x, y, \mathbf{t_o}) \leftarrow v_2(x, y).$

$\quad P\_(x, y, \mathbf{nv_1}) \leftarrow P\_(x, y, \mathbf{t_o}).$

$\quad P\_(x, y, \mathbf{nv_2}) \leftarrow P\_(x, y, \mathbf{v_1}).$

$\quad P(x, y) \leftarrow P\_(x, y, \mathbf{v_1}).$

$$P(x, y) \leftarrow P_{-}(x, y, \mathbf{t_o}).$$

$$R(x, y) \leftarrow R_{-}(x, y, \mathbf{v_1}).$$

$\Pi(\mathcal{R}, IC):$

$$P_{-}(x, y, \mathbf{t^\star}) \leftarrow P_{-}(x, y, \mathbf{t_a}).$$

$$P_{-}(x, y, \mathbf{t^\star}) \leftarrow P(x, y).$$

$$R_{-}(x, y, \mathbf{t^\star}) \leftarrow R_{-}(x, y, \mathbf{t_a}).$$

$$R_{-}(x, y, \mathbf{t^\star}) \leftarrow R_{-}(x, y, \mathbf{t_d}).$$

$$R_{-}(x, y, \mathbf{f_a}) \vee R_{-}(y, x, \mathbf{t_a}) \leftarrow R_{-}(x, y, \mathbf{t^\star}), R_{-}(y, x, \mathbf{f_a}), dom(x), dom(y).$$

$$R_{-}(x, y, \mathbf{f_a}) \vee R_{-}(y, x, \mathbf{t_a}) \leftarrow R_{-}(x, y, \mathbf{t^\star}), \ not \ R(y, x), dom(x), dom(y).$$

$$P_{-}(x, y, \mathbf{t^{\star\star}}) \leftarrow P_{-}(x, y, \mathbf{t^\star}), \ not \ P_{-}(x, y, \mathbf{f_a}).$$

$$R_{-}(x, y, \mathbf{t^{\star\star}}) \leftarrow R_{-}(x, y, \mathbf{t^\star}), \ not \ R_{-}(x, y, \mathbf{f_a}).$$

$$\leftarrow R_{-}(x, y, \mathbf{t_a}), R_{-}(x, y, \mathbf{f_a}).$$

$$\leftarrow P_{-}(x, y, \mathbf{t_a}), P_{-}(x, y, \mathbf{f_a}). \qquad \square$$

Definition 6.14 The global instance associated to a choice model $\mathcal{M}$ of $\Pi(\mathcal{G}, IC)$ is $D_{\mathcal{M}} = \{P(\bar{a}) \mid P \in \mathcal{R} \text{ and } P(\bar{a}, \mathbf{t^{\star\star}}) \in \mathcal{M}\}.$ $\qquad \square$

Example 6.27 (example 6.20 and 6.26 continued) Program $\Pi(\mathcal{G}, IC) = \Pi(\mathcal{G}) \cup \Pi(\mathcal{R}, IC)$ has five stable models with the following associated repairs: (a) $D_{\mathcal{M}_1^r} = \{P(a, b), R(b, b), P(a, c)\}$, corresponding to the already consistent minimal instance $D_{\mathcal{M}_1}$ in Example 6.21; (b) $D_{\mathcal{M}_2^r} = \{P(a, a), P(a, c)\}$ and $D_{\mathcal{M}_3^r} = \{R(a, b), R(b, a), P(a, a), P(a, c)\}$, the repairs of the inconsistent instance $D_{\mathcal{M}_2}$; (c) $D_{\mathcal{M}_4^r} = \{P(a, c)\}$ and $D_{\mathcal{M}_5^r} = \{R(c, b), R(b, c), P(a, c)\}$, the repairs of instance $D_{\mathcal{M}_3}$; and (d) $D_{\mathcal{M}_6^r} = \{P(a, u), P(a, c)\}$ and $D_{\mathcal{M}_7^r} = \{R(u, b), R(b, u), P(a, u), P(a, c)\}$, the repairs of $D_{\mathcal{M}_4}$. $\qquad \square$

By construction, the repair program can be *split* [Lifschitz and Turner, 1994] into the specification of the minimal instances, followed by the specification of their repairs. Therefore, the minimal legal instances can be computed first, and next, the repairs
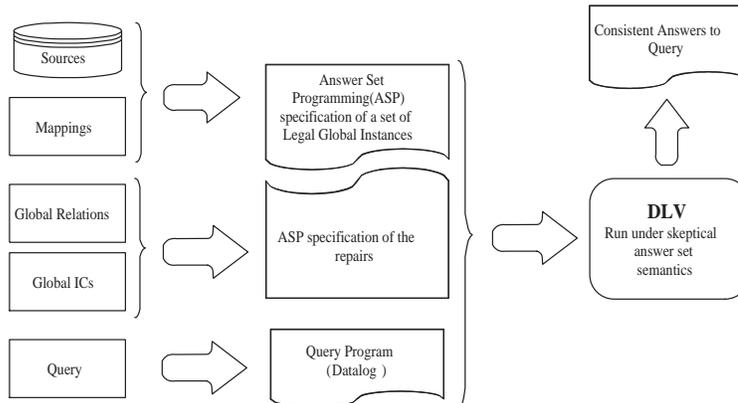
Figure 6.2: Computing consistent answers

of them. Each minimal model computed by the first part of $\Pi(\mathcal{G}, IC)$ can be seen as a simple, relational database, which is repaired afterwards by the second part of $\Pi(\mathcal{G}, IC)$. This gives us the following theorem.

Theorem 6.4 Let IC be an RIC-acyclic set of UICs and RICs. If $\mathcal{M}$ is a choice model of $\Pi(\mathcal{G}, IC)$, then $D_{\mathcal{M}}$ is a repair of $\mathcal{G}$ with respect to IC. Furthermore, the repairs obtained in this way are all the repairs of $\mathcal{G}$ with respect to IC. $\square$

In the case of cyclic sets of RICs, IC, the global instances associated to the choice models of the program will always be a superset of the repairs of $\mathcal{G}$ with respect to IC, and in order to obtain the repairs, the choice models will have to be compared, to choose those that minimally differ from the minimal legal instance.

6.4   Consistent Answers

To obtain those answers to a query posed on a DIS $\mathcal{G}$ that are consistent with respect to IC, we can run the program $\Pi(Q) \cup SV(\Pi(\mathcal{G}, IC))$ and get the answer from the intersection of the stable models, as in the case of stand alone databases (see Section 5.2.1). Figure 6.2 describes the methodology in general terms.

Example 6.28 (example 6.20, 6.26 and 6.27 continued) We want the consistent an-swers to the query $Q : P(x, y)$. First, the query is written as the query program clause $Ans(x, y) \leftarrow P_{\cdot}(x, y, \mathbf{t}^{\star\star})$. This query program, $\Pi(Q)$, is run with $SV(\Pi(\mathcal{G}_3, sim))$. The corresponding stable models of $\Pi(Q) \cup SV(\Pi(\mathcal{G}_3, sim))$ are: (a) $\overline{\mathcal{M}}_1^r = \mathcal{M}_1^r \cup \{Ans(a, b), Ans(a, c)\}$; (b) $\overline{\mathcal{M}}_2^r = \mathcal{M}_2^r \cup \{Ans(a, a), Ans(a, c)\}$; $\overline{\mathcal{M}}_3^r = \mathcal{M}_3^r \cup \{Ans(a, a), Ans(a, c)\}$; (c) $\overline{\mathcal{M}}_4^r = \mathcal{M}_4^r \cup \{Ans(a, c)\}$; $\overline{\mathcal{M}}_5^r = \mathcal{M}_5^r \cup \{Ans(a, c)\}$; (d) $\overline{\mathcal{M}}_6^r = \mathcal{M}_6^r \cup \{Ans(a, u), Ans(a, c)\}$; $\overline{\mathcal{M}}_7^r = \mathcal{M}_7^r \cup \{Ans(a, u), Ans(a, c)\}$. $Ans(a, c)$ is the only query atom in all stable models, then the tuple (a, c) is the only consistent answer to the query. □

If $\mathcal{G}$ is consistent, then the consistent answers to Q computed with this method coincide with the minimal answers to Q, and then to the certain answers if Q is monotone.

6.5 Further Analysis, Extensions and Discussion

6.5.1 Complexity

The complexity analysis of CQA in integration of open sources under the LAV ap-proach can be split according to the main two layers of the combined program, namely, the specification of minimal instances and the specification of the repairs of those min-imal instances.

Query evaluation from the program $\Pi(\mathcal{G})$ with choice under the skeptical stable model semantics is in *coNP* (the case singularized as *certainty semantics* in [Wang and Zaniolo, 2000]). Actually, if the choice operator program is represented in its "classi-cal" stable version (see Section 6.2.1), we are left with a normal (non-disjunctive), but non-stratified program whose query answering complexity under the skeptical stable model semantics is *coNP*-complete [Dantsin *et al.*, 2001; Leone *et al.*, 2006] in data

[Abiteboul *et al.*, 1995]. In our case, this means in terms of the combined sizes of the sources.

This complexity of computing minimal answers is inherited by the computation of certain answers when the two notions coincide, e.g., for monotone queries, like Datalog queries. This complexity result is consistent and matches the theoretical complexity lower bound on computing certain answers to Datalog queries under the LAV approach [Abiteboul and Duschka, 1998]. With disjunctive views, as considered in Section 6.5.4, the complexity of the program goes up to Π_2^P-complete.

The complexity of query evaluation with respect to the disjunctive normal program $\Pi(\mathcal{G}, IC)$ that specifies the repair of minimal instances is Π_2^P-complete in data complexity [Dantsin *et al.*, 2001], which matches the complexity of consistent query answering [Bertossi and Chomicki, 2003; Chomicki and Marcinkowski, 2005a; Calì *et al.*, 2003a].

In the cases in which the repair part of the program for CQA is *head-cycle free* (HCF) (see Section 5.2.3), the complexity is reduced to *coNP* [Ben-Eliyahu and Dechter, 1994; Leone *et al.*, 1997]. The program $\Pi(\mathcal{G}, IC)$ is HCF for a combination of: (a) *Denial constraints*, i.e., formulas of the form $\bigvee_{i=1}^n P_i(\bar{t}_i) \rightarrow \varphi$, where $P_i(\bar{t}_i)$ is an atom, and φ is a formula containing built-in predicates only; (b) *Acyclic referential integrity constraints*, i.e., without cycles in the dependency graph. This case includes the usual integrity constraints found in database practice, like (non-cyclic) foreign key constraints.

6.5.2 Infinite vs. Finite Domain

In Section 6.1.4, we considered the possibility of having an infinite underlying domain $\mathcal{U}$. At the purely specification level there is no problem in using, in the first item of Definition 6.8, an infinite number of facts. Our soundness and completeness theorems

hold. However, in the logic programs we have presented in the examples we had a finite domain, see Example 6.19 (the finite domain is specified by the *dom* predicate), but also an extra constant u that does not appear in the active domain (all the constants in the sources plus those that appear in the view definitions) of the integration system.

The reason is that we need a finite domain to run the program, but at the same time, we need to capture the potential infiniteness of the domain and the openness of the sources. Furthermore, we should not be forced to use only the active domain, because doing so might assign the wrong semantics to the integration system.

Example 6.29 Consider an integration system $\mathcal{G}_4$ with one source defined by the view $V(x) \leftarrow R(x,y)$, and the query $Q(y) \leftarrow R(x,y)$. If the view extension has only one tuple, say $\{(a)\}$, then the active domain is $\{a\}$ and $R(a,a)$ is in all the legal instances of $\mathcal{G}_4$ if only this domain is used; and we would have $Certain_{\mathcal{G}_4}(Q) = \{(a)\}$. Now, if the view extension becomes $\{(a),(b)\}$, the active domain is $\{a,b\}$, and there is a global instance containing just the tuple $R(a,b)$, and another containing just $R(a,a)$. In consequence, there will be no certain answers. This simple example shows that a positive query may have an undesirable non-monotonic behavior $\qquad\square$

In Example 6.19, introducing one extra constant (u) is good enough to correctly answer conjunctive queries (see below). In the general case, the number of extra constants may vary depending on the situation.

It is necessary to make all these considerations, because the set of minimal legal instances may depend on an underlying domain, as we saw in Example 6.13, where

$Mininst(\mathcal{G}_2) = \{\{P(a,c),\, P(a,z),\, R(z,a)\} \mid z \in \mathcal{U} = \{a,b,c,...\}\}$.

Since we want only the certain answers, those that can be obtained from all the stable models, it is easy to see that the values taken by the "free variables", like z above, will not appear in a certain answer. However, the absence of the extra,

new constants may sanction as certain some answers that are not if the domain is restricted to the active domain (see Example 6.29). As a consequence, we need a larger domain, with enough values to represent the relationships and differences between the free variables.

Depending on the query, there is a finite domain that generates the same certain and minimal answers as the infinite domain. It can be shown that if the query is conjunctive, then adding only one new constant to the active domain is good enough (see Example 6.19).

If the query is disjunctive, then the smallest "equivalent" finite domain is the active domain plus n new constants, where n is the maximum number of instantiations of existential variables in a minimal legal instance. This number of instantiations cannot be obtained from the view definitions alone, because it also depends on the number of elements in the sources associated to the Skolem predicates. An upper bound on the number of constants to be added to the active domain to correctly answer disjunctive queries is the sum over all sources of the product of the number of existential variables in a view definition with the number of atoms in the corresponding source.

Example 6.30 Consider an integration system $\mathcal{G}_5$:

$$V_1(x, y) \leftarrow P(x, z_0), R(z_0, y); \qquad \{V_1(a, b)\}.$$
$$V_2(x, y) \leftarrow P(x, z_1), R(z_2, y); \qquad \{V_2(a, b), V_2(c, d)\}.$$

The set of minimal legal instances is $\{\{P(a, z_1), R(z_1, b), P(c, z_2), R(z_3, d)\} \mid z_1, z_2, z_3 \in \mathcal{U}\}$. By looking at this representation, we see that in order to obtain correct certain answers to disjunctive queries, it is sufficient to add three extra constants to the active domain $\{a, b, c, d\}$, obtaining, say $\mathcal{U} = \{a, b, c, d, e, f, g\}$, a finite domain that is able to simulate an infinite domain with respect to disjunctive queries.

Instead of inspecting the minimal instances to determine the number of new constants, we can use an upper bound, in this case, five, which can be computed as: 1

existential variable times 1 atom plus 2 existential variables times 2 atoms. So, we could use a domain $\mathcal{U}$ with five extra constants. $\qquad\qquad\qquad\qquad\qquad\qquad$ □

6.5.3 Choice Models vs. Skolem Functions

In this chapter, we have used the *choice* operator to replace the Skolem functions used in the inverse rules algorithm. In this way, we were able to specify the minimal global instances. This was one of our original goals, and is interesting in itself. This result allows us to specify the repairs of the integration system with respect to the ICs. However, if we are interested in query answering only, it becomes relevant to analyze if it is possible to retrieve the minimal, certain and consistent answers by keeping the Skolem functions in the program, evaluating the program, and then filtering out the final answers that contain those functions (as done in [Duschka *et al.*, 2000] to obtain certain answers).

We first analyze the case of the simple program (see Section 6.2.1), in which we want to consider using Skolem functions instead of the functional predicate together with the choice operator. For example, we would have $P(x, f(x)) \leftarrow V(x)$, instead of the rules $P(x, y) \leftarrow V(x), F(x, y)$ and $F(x, y) \leftarrow V(x), dom(y), choice((x), (y))$.

In this case, the program will have the same rules $\mathcal{V}^{-1}$ as in the inverse rules algorithm. The resulting definite program is positive and, therefore, its stable model corresponds to the minimal model. That model will have atoms with instantiated Skolem functions, and can be seen as a compact representation of the collection of stable models of the choice program, in the sense that the latter can be recovered by considering the different ways in which the Skolem functions can be defined in the underlying domain.

If a query is posed using the program with Skolem functions, the set of answers may or may not contain tuples with ground Skolem functions. Those answers with

Skolem functions correspond to answers that would be different in different stable models of the choice program, because in a sufficiently rich domain (see Section 6.5.2) the functions may be defined in different ways. This is why if we delete those answers with functions, we get the same answers as from the choice program $\Pi(\mathcal{G})$ under the cautious stable model semantics. As a consequence, for computing the certain answers to a monotone query, we can use either the program with Skolem functions (pruning the answers with Skolem functions at the end) or the choice program.

Let us now consider the refined program (see Section 6.2.2). In this case, if Skolem functions are used instead of the choice operator, the resulting program is a normal program that may have several stable models.

Example 6.31 Consider an integration system $\mathcal{G}$ with

$$V_1(x) \leftarrow P(x_1, y_1, z_1), S(y_1) \qquad V_1(a)$$
$$V_2(x, y) \leftarrow P(x_2, y_2, z_2) \qquad V_2(a, e)$$

The following is the program with Skolem functions:

$P_{\_}(x, f_1(x), f_2(x), \mathbf{v_1}) \leftarrow add_{v_1}(x), add_{v_1 y}(x), add_{v_1 z}(x).$

$S_{\_}(f_1(x), \mathbf{v_1}) \leftarrow addv_1(x).$

$add_{v_1}(x) \leftarrow v_1(x), \ not \ aux_{v_1}(x).$

$aux_{v_1}(x) \leftarrow var_{v_1 y}(x, y, z), var_{v_1 z}(x, y, z).$

$var_{v_1 y}(x, y, z) \leftarrow P_{\_}(x, y, z, \mathbf{nv_1}), S_{\_}(y, \mathbf{nv_1}).$

$var_{v_1 z}(x, y, z) \leftarrow P_{\_}(x, y, z, \mathbf{nv_1}).$

$add_{v_1 y}(x) \leftarrow add_{v_1}(x), \ not \ aux_{v_1 y}(x).$

$aux_{v_1 y}(x) \leftarrow var_{v_1 y}(x, y, z), z = f_2(x).$

$add_{v_1 z}(x) \leftarrow add_{v_1}(x), \ not \ aux_{v_1 z}(x).$

$aux_{v_1 z}(x) \leftarrow var_{v_1 z}(x, y, z), z = f_1(x).$

$P_{\_}(x, y, f_3(x, y), \mathbf{v_2}) \leftarrow add_{v_2}(x, y), add_{v_2 z}(x, y).$

$add_{v_2}(x, y) \leftarrow v_2(x, y), \ not \ aux_{v_2}(x, y).$

$aux_{v_2}(x, y) \leftarrow var_{v_2 z}(x, y, z).$

$var_{v_2 z}(x, y, z) \leftarrow P\_(x, y, z, \mathbf{nv_2}).$

$add_{v_2 z}(x, y) \leftarrow add_{v_2}(x, y), \ not \ aux_{v_2 z}(x, y).$

$aux_{v_2 z}(x, y) \leftarrow var_{v_2 z}(x, y, z).$

$P\_(x, y, z, \mathbf{nv_1}) \leftarrow P\_(x, y, z, \mathbf{v_2}).$

$P\_(x, y, z, \mathbf{nv_2}) \leftarrow P\_(x, y, z, \mathbf{v_1}).$

$P(x, y, z) \leftarrow P\_(x, y, z, \mathbf{v_1}).$

$P(x, y, z) \leftarrow P\_(x, y, z, \mathbf{v_2}).$

$S(y) \leftarrow S\_(y, \mathbf{v_1}).$ $\hfill \square$

The stable models of the refined program with Skolem functions are computed under the *unique names assumption* [Reiter, 1984]. As a consequence of this, the program may not be able to distinguish those cases where the openness condition for a source can be satisfied because the condition already holds for another source (see the discussion at the end of Section 6.2.1). For example, if two atoms, say $P(a, f_1(a), f_2(a))$ and $P(a, e, f_3(a, e))$, are added to the stable models in order to satisfy the openness conditions for two different views, the program will treat those two atoms as different. This may not be the case when the Skolem functions are interpreted. As a consequence, stable models that are larger than needed might be produced.

If each of these stable models is seen as a compact representation of a set of intended global instances, which can be recovered through all possible instantiations of the Skolem functions in the model, we may end up generating global instances that are not minimal. In other words, the class of stable models of the refined program with Skolem functions represents a class that possibly properly extends the one of minimal instances, by including global instances that are legal, but not minimal.

Example 6.32 (example 6.31 continued) The minimal instances of this integration system can be represented by $\{\{P(a, e, f_3(a, e)), P(a, f_1(a), f_2(a)), S(f_1(a))\} \mid$

$f_3(a,e) \in \mathcal{U}$, $f_2(a) \in \mathcal{U}$, $f_1(a) \in \mathcal{U} \smallsetminus \{e\}\} \bigcup \{\{P(a,e,f_3(a,e)),\ S(e)\} \mid f_3(a,e) \in \mathcal{U}\}$.

By interpreting the Skolem functions in the underlying domain, we obtain all and only the minimal instances. Notice that in this case, it is necessary to give all the possible values in the domain to the existential variables (or function symbols), the only exception being when the existential variable y_1 is made equal to e. In that case it is good enough to give values to z_1 *or* z_2 in order to satisfy the openness conditions for V_1 and V_2.

In the context of the refined program with function symbols, due to the unique names assumption, $f_1(a)$ will always be considered different from e, and therefore the program will not realize that there is a minimal model that does not contain the tuple $P(x, f_1(x), f_2(x), v_1)$. As a consequence, the program will generate the stable model $\{P(a,e,f_3(a,e)),\ P(a,f_1(a),f_2(a)),\ S(f_1(a))\}$, that represents a proper superclass of the minimal legal instances. For example, it represents the instance $\{P(a,e,u), P(a,e,v), S(e)\}$, that is not minimal. □

The possibly strict superset of the minimal instances that is represented by the models of the program with functions can be used to correctly compute the minimal and certain answers to monotone queries (in this case it is better to use the simple program though), but not for queries with negation.

We now consider the repair program. In those cases where the stable models of the simple or revised programs with Skolem functions do not represent the minimal legal instances, it is clear that it is not possible to compute their repairs. When the stable models do represent the minimal legal instances, it is not possible for the repair program to detect all the inconsistencies in them because of the underlying unique names assumption.

Example 6.33 (examples 6.12 and 6.13 continued) The minimal legal instances are represented via Skolem functions by $\mathcal{M} = \{P(a,f(a,b)), R(f(a,b),b), P(a,c)\}$, which

can be obtained as a model of the simple program with Skolem functions. This model
is inconsistent with respect to $IC\colon \forall x\forall y(R(x,y) \to R(y,x))$.

The repair program $\Pi(\mathcal{G}, IC)$ has the rule

$$R(x, y, \mathbf{f_a}) \vee R(y, x, \mathbf{t_a}) \leftarrow R(x, y, \mathbf{t^\star}), R(y, x, \mathbf{f^\star}),$$

that will produce the repairs $D_{\mathcal{M}_1} = \{P(a, f(a,b)), P(a,c)\}$ and $D_{\mathcal{M}_2} = \{P(a, f(a,b)),$
$R(f(a,b),b), R(b, f(a,b)), P(a,c)\}$, which represent a superset of the real repairs of
the minimal legal instances. Because of the unique names assumption, the program
will not detect that, for $f(a,b) = b$, the instance is consistent with respect to IC. $\square$

6.5.4 Disjunctive Sources

In Section 6.2, we considered sources defined as conjunctive views only. If sources are
now described as disjunctive views, i.e., with more than one conjunctive rule [Duschka,
1997], then the program $\Pi(\mathcal{G})$ has to be extended in order to capture the minimal
instances. In this case, a *source* S_i is a pair $\langle \Phi_i, v_i \rangle$, where Φ_i is a set of conjunctive
rules defining the same view, say $\varphi_{i1}, \dots, \varphi_{im}$, and v_i is the given extension of the
source.

Definition 6.15 Given an open global system $\mathcal{G} = \{\langle \Phi_1, v_1 \rangle, \dots, \langle \Phi_n, v_n \rangle\}$, the set
of legal global instances is $\quad Linst(\mathcal{G}) = \{D$ instance over $\quad \mathcal{R} \mid \quad v_i \subseteq \bigcup_k \varphi_{ik}(D),$
for $\quad i = 1, \dots, n\}$. $\square$

Example 6.34 Consider the global integration system $\mathcal{G}_7$ with global relations $\{R(x,y),$
$S(x), T(x,y)\}$ and two source relations v_1 and v_2 with the view definitions and ex-
tensions shown in Table 6.2.

Examples of legal instances are $\{S(b), S(a), R(a,b), T(c,d)\}$, $\{S(b), S(a), R(a,b),$
$R(c,d), S(d)\}$ and $\{S(b), S(a), R(a,b), T(c,d), T(a,b)\}$. $\square$

Source	Extension	View Definitions
v_1	$\{V_1(a,\,b),\,V_1(c,\,d)\}$	$\mathcal{V}_{11}:\quad V_1(x,y) \leftarrow R(x,y), S(y)$
		$\mathcal{V}_{12}:\quad V_1(x,d) \leftarrow T(x,d)$
v_2	$\{V_2(b), V_2(a)\}$	$\mathcal{V}_{21}:\quad V_2(x) \leftarrow S(x)$

Table 6.2: View definitions and extension of Example 6.34

If we have disjunctive view definitions, in order to satisfy the openness of a source, it is necessary that one or more views generate each of its tuples. To capture this, in [Duschka, 1997] the concepts of *truly disjunctive* view and *witness* are introduced, together with an *exclusion condition*. Informally, a set of views is *truly disjunctive* if there is a tuple $\bar{t}$ that can be generated by any of the views. This tuple is called a *witness*. The *exclusion condition* is a constraint on the *witness* that determines for which tuples the *truly disjunctive* views are the most general.

Example 6.35 (example 6.34 continued) The atoms of v_1 that have the constant d as the second attribute can be generated either by $\mathcal{V}_{11}$ or $\mathcal{V}_{12}$. On the other hand, if the second attribute is different from d, the atom can only be generated by $\mathcal{V}_1$. This is expressed in terms of truly disjunctive views, most general witness and exclusion condition that are shown in Table 6.3. $\square$

truly disjunctive views	most general witness	exclusion condition
$\mathcal{V}_1$	(x_1, x_2)	second attribute $\neq d$
$\mathcal{V}_1, \mathcal{V}_2$	(x_1, d)	true

Table 6.3: Truly disjunctive views of Example 6.35

In order to extend the simple version of $\Pi(\mathcal{G})$, incorporating disjunctive view definitions, we need to take into account the different sets of truly disjunctive views with their witnesses and exclusion conditions. For example, for the second truly disjunctive set in Example 6.35, the following rule needs to be imposed

$$(R(x,d) \wedge S(d)) \vee T(x,d) \leftarrow V(x,d), \tag{6.4}$$

which is equivalent to the pair of disjunctive Datalog rules

$$R(x, d) \vee T(x, d) \quad \leftarrow \quad V(x, d) \tag{6.5}$$

$$S(d) \vee T(x, d) \quad \leftarrow \quad V(x, d). \tag{6.6}$$

For each set of truly disjunctive views, rules like (6.5) and (6.6) will have to be satisfied by the legal instances. These remarks motivate the following program as an specification of the minimal legal instances.

Definition 6.16 Given an open global system $\mathcal{G}$, the program, $\Pi^{\vee}(\mathcal{G})$, contains the following clauses:

1. Fact $\quad dom(a)$, for every constant $a \in \mathcal{U}$. Also the fact $V_i(\bar{a})$ whenever $\bar{a} \in v_i$ for some source extension v_i in $\mathcal{G}$.

2. For every set of truly disjunctive views for a source V_i of the form

$$\mathcal{V}_{i1}: \quad V_i(\bar{x}_1) \leftarrow P_{11}(\bar{x}_{11}), \ldots, P_{1n}(\bar{x}_{1n_1})$$

$$\cdots$$

$$\mathcal{V}_{ik}: \quad V_i(\bar{x}_k) \leftarrow P_{k1}(\bar{x}_{k1}), \ldots, P_{kn}(\bar{x}_{kn_k}),$$

where the variables in each view are different (fresh), for its more general witness $\bar{w}$ and its most general exclusion condition φ, the rules

$$P_{1\delta_1}(\bar{x}'_{1\delta_1}) \vee \cdots \vee P_{k\delta_k}(\bar{x}'_{k\delta_k}) \leftarrow V_i(\bar{w}) \wedge \varphi \wedge \bigwedge_{z_l \in (\bar{x}' \backslash \bar{w})} F_i^l(\bar{w}, z_l),$$

where $\bar{x}' = \bigcup_{j=1}^{k} \bar{x}'_{j\delta_j}$, and $\delta_l \in \{1, \ldots, n_k\}$ for $l = 1, \ldots, k$.

The tuples of variables $\bar{x}'_{1\delta_1}, \ldots, \bar{x}'_{k\delta_k}$ are those obtained by the substitution of $\bar{x}_i$ by $\bar{w}$ in all the view definitions. These rules represent all the possible combinations of k predicates, where each of them is chosen from a different view definition.

3. For every predicate $F_i^l(\bar{x}, z_l)$ introduced in 2., the rule

$$F_i^l(\bar{x}, z_l) \leftarrow V_i(\bar{x}), dom(z_l), choice((\bar{x}), (z_l)). \qquad \square$$

Example 6.36 (example 6.35 continued) The program $\Pi^\vee(\mathcal{G}_7)$ is:

1. $dom(a). \quad dom(b). \quad dom(c). \quad dom(d).$

2. $R(x, y) \leftarrow V_1(x, y), y \neq d.$

3. $S(y) \leftarrow V_1(x, y), y \neq d.$

4. $T(x, d) \vee R(x, y) \leftarrow V_1(x, y).$

5. $T(x, d) \vee S(y) \leftarrow V_1(x, y).$

6. $S(x) \leftarrow V_2(x).$

Rules (2)-(3) and (4)-(5) represent, respectively, the first and second truly disjunctive set for source v_1. Rule (6) is for the non-disjunctive source v_2. $\qquad \square$

If all the sources are defined by conjunctive views, then is easy to see that $\Pi^\vee(\mathcal{G})$ becomes the simple program $\Pi(\mathcal{G})$ introduced in Section 6.2.1. As before, it holds that

$$Mininst(\mathcal{G}) \subseteq \{D_\mathcal{M} \mid \mathcal{M} \text{ is a stable model of } \Pi^\vee(\mathcal{G})\} \subseteq Linst(\mathcal{G}).$$

For monotone queries Q, the answers obtained using $\Pi^\vee(\mathcal{G})$ coincide with $Certain_\mathcal{G}(Q)$ and $Minimal_\mathcal{G}(Q)$. This might not be the case of queries with negation. It is possible to give a refined version for the case of disjunctive views, corresponding to the non-disjunctive program in Section 6.2.2, for which $Mininst(\mathcal{G}) = \{D_\mathcal{M} \mid \mathcal{M} \text{ is a stable model of } \Pi^\vee(\mathcal{G})\}$ also holds.

6.5.5   The Mixed Case

So far we have assumed that all the sources are open. Now we will consider the *mixed case*, where some of the sources may be *closed* or closed and open (*clopen* or

exact) [Grahne and Mendelzon, 1999]. Intuitively speaking, a closed source contains a superset of the data of its kind in the system, and the clopen source contains exactly all the data of its kind in the system.

More precisely, if a material source relation v, defined as the view $V(\bar{x}) \leftarrow \varphi_V(\bar{x})$ of the global system, has been defined as a closed (clopen) source, then in any legal instance D, it must hold $v \supseteq \varphi_V(D)$ (resp. $v = \varphi_V(D)$).

In this section, we will describe how to modify the program that specifies the minimal instances presented in Section 6.2 when some of the sources are declared closed or clopen.

Example 6.37 For the domain $\mathcal{U} = \{a, b, c, \ldots\}$, consider the integration system $\mathcal{G}_4$:

$$V_1(x, z) \leftarrow P(x, y), R(y, z); \qquad v_1 = \{(a, b)\} \quad open.$$
$$V_2(x, y) \leftarrow P(x, y); \qquad\qquad v_2 = \{(a, c)\} \quad clopen.$$

In Example 6.13, we had the same sources and definitions, but then they were all declared open; and $Mininst(\mathcal{G}_2) = \{\{P(a, c), P(a, z), R(z, b)\} \mid z \in \mathcal{U}\}$. Now, the label on the second sources forces relation P to be $\{(a, c)\}$. As a consequence, we obtain $Mininst(\mathcal{G}_4) = \{\{P(a, c), R(c, b)\}\}$. □

It is clear that the closed and clopen labels will impose additional restrictions on the legal instances we had for the purely open case, when all sources are open. In particular, these labels will never force to add new tuples to the legal instances. Actually, if a source is declared closed, then that source will contribute with the empty set of tuples to the minimal instances of the integration system.

With open, closed and clopen sources, the sets of legal and minimal instances will always be subsets of the same sets for the case where the same sources are all declared open. In order to obtain the minimal instances in the mixed case, all we

have to do is filter out some of the minimal instances obtained in the purely open case, namely those that violate the closedness condition for some of the sources. This can be captured at the logic program specification level by means of a program denial constraint, which has the effect of discarding some of the stable models.

In the mixed case, the program $\Pi_{mix}(\mathcal{G})$ that specifies the minimal instances consists of the program $\Pi(\mathcal{G})$ as defined in Section 6.2 (as if all the sources were open) plus a denial constraint of the form $\leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n), \ not \ V(\bar{x})$, for each closed (or clopen) source v with view definition $V(\bar{x}) \leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$. That is, the open sources contribute with rules to the program, the clopen sources both with rules and program denial constraints, and the closed sources with program denial constraints only.

With these modifications, the obtain the same correspondence between the stable models of the program $\Pi_{mix}(\mathcal{G})$ and the minimal instances of the mixed integration system $\mathcal{G}$.

Example 6.38 (example 6.37 continued) The program $\Pi_{mix}(\mathcal{G}_4)$ that specifies the minimal instances of system $\mathcal{G}_4$ is:

1. $dom(a)$. $dom(b)$. $dom(c)$. $dom(u)$. $V_1(a, b)$. $V_2(a, c)$.

2. $P(x, z) \leftarrow V_1(x, y), F_1(x, y, z)$.
   $R(z, y) \leftarrow V_1(x, y), F_1(x, y, z)$.
   $P(x, y) \leftarrow V_2(x, y)$.

3. $F_1(x, y, z) \leftarrow V_1(x, y), dom(z), choice((x, y), (z))$.

4. $\leftarrow P(x, y), \ not \ V_2(x, y)$.

This program, excluding the last program denial constraint, coincides with program $\Pi(\mathcal{G}_2)$ in Example 6.19, where the same sources and definitions are considered, but all

the sources are open only. With the denial constraint, that enforces the closeness of source V_2, the only stable model of $\Pi_{mix}(\mathcal{G}_4)$ is $\{dom(a), \dots, V_1(a, b), V_2(a, c), \underline{P(a, c)},$ $F_1(a, b, c), \underline{R(c, b)}\}$, which corresponds to the only minimal instance $\{\{P(a, c), R(c, b)\}\}$. $\qquad\square$

Notice that the solution we have reached via logic programs is similar in spirit to the solution presented in [Grahne and Mendelzon, 1999], where the mixed case is treated. There, tableaux with constraints are used to compactly represent the legal instances and obtain certain answers. The tableaux capture the open part, and the constraints, as in our solution, the closed part.

6.5.6 GAV mappings

So far we have addressed CQA in data integration systems under the LAV approach. Here, we will extend those results to the GAV approach. First we need some definitions.

In the GAV approach, as presented at the beginning of this chapter, a *view* is of the form $R(\bar{t}) \leftarrow body(\varphi_R)$, where φ_R is a conjunctive query over $\mathcal{S}$-atoms, $body(\varphi_R)$ is the body of φ_R, and $R \in \mathcal{R}$. An open GAV system is defined by $\langle \mathcal{R}, \mathcal{S}, \mathcal{V} \rangle$ where $\mathcal{R}$ is the global schema, $\mathcal{S}$ the set of open sources, and $\mathcal{V}$ the set of views.

Definition 6.17 Given a GAV global system $\mathcal{G} = \langle \mathcal{R}, \mathcal{S}, \mathcal{V} \rangle$, the set of legal global instances is $Linst(\mathcal{G}) = \{D \text{ instance over } \mathcal{R} \mid R(D) \supseteq \varphi_R(\mathcal{S}) \text{ for } (R(\bar{t}) \leftarrow body(\varphi_R)) \in \mathcal{V}\}$. The retrieved database, $Ret(\mathcal{G}) = \{A \mid A \in \varphi_R(\mathcal{S}), \text{ for } \varphi_R \in \mathcal{V}\}$. $\qquad\square$

Example 6.39 Consider source relations $S_1 = \{(a, b), (a, e)\}$ and $S_2 = \{(b, a), (b, f)\}$, the global schema $\mathcal{R} = \{R_1, R_2\}$, and the view definitions:

$R_1(x, z) \leftarrow S_1(x, y), S_2(y, z),$

$R_2(x, y) \leftarrow S_1(x, y), S_2(z, x).$

Here, $Ret(\mathcal{G}) = \{R_1(a,a), R_1(a,f), R_2(a,b)\}$, and $Linst(\mathcal{G}) = \{D \mid D$ has schema $\mathcal{R}$, and $D \supseteq \{R_1(a,a), R_1(a,f), R_2(a,b)\}$. $\qquad\square$

It is easy to see that the set of legal instances of a GAV integration system are the databases that are a superset of the retrieved database.

The retrieved database corresponds to what is obtained by directly applying the view definitions to the sources. The definition of certain answer, minimal legal instance, consistency and minimal answers are the same as under LAV (see Definitions 6.2, 6.3 and 6.4).

Example 6.40 (example 6.39 continued) The minimal legal instance is unique and is equal to the retrieved database. The certain answers to the query $Q(x) \leftarrow R_1(x,y)$, $R_2(x,z)$ are $Certain(Q) = \{(a)\}$. Since the query is positive, the minimal answers coincide with the certain answers. Now, if we add an IC: $\forall xyz(R_1(x,y) \wedge R_2(x,z) \rightarrow y = z)$, the integration system is inconsistent, since the minimal legal instance does not satisfy the constraint. $\qquad\square$

In the GAV case there is always a unique minimal legal instance, that corresponds to the retrieved database.

Definition 6.18 Given an open global system $\mathcal{G}$, the program $\Pi^{GAV}(\mathcal{G})$ contains the rule $R(\bar{x}) \leftarrow S_1(\bar{x}_1), \ldots, S_n(\bar{x}_n)$, for each view definition $(R(\bar{x}) \leftarrow S_1(\bar{x}_1), \ldots, S_n(\bar{x}_n)) \in \mathcal{V}$. $\qquad\square$

Definition 6.19 The database associated to a model $\mathcal{M}$ of $\Pi^{GAV}(\mathcal{G})$ is $D_\mathcal{M} = \{P(\bar{a}) \mid P(\bar{a}) \in \mathcal{M}$ and $P \in \mathcal{R}\}$. $\qquad\square$

Proposition 6.1 The program $\Pi^{GAV}(\mathcal{G})$ has a unique minimal model $\mathcal{M}$ (which is also a unique stable model), such that $D_\mathcal{M} = Ret(\mathcal{G})$. $\qquad\square$

This implies that program $\Pi^{GAV}(\mathcal{G})$ can be used to retrieve the certain answers for monotone queries and the minimal answers for any type of Datalog$^{\neg,\vee}$ queries.

A GAV data integration system can be said to be consistent if the minimal legal instance, i.e., the retrieved legal instance, satisfies the global constraints. The formal definitions of repairs and CQA are the same as the one given for LAV (see Definition 6.6 and 2.5). Therefore, the repairs of the integration system will be, in the GAV case, the set of repairs of the retrieved database.

Definition 6.20 Given an integration system $\mathcal{G}$ with global schema $\mathcal{R}$ and a set of global ICs IC, the *repair program* is:

$$\Pi(\mathcal{G}, IC) \ := \ \Pi^{GAV}(\mathcal{G}) \ \cup \ \Pi(\mathcal{R}, IC).$$

□

$\Pi(\mathcal{R}, IC)$ consists of the intentional part of the repair program $\Pi(D, IC)$.

This repair program can be used to retrieve the consistent answers from a GAV data integration system.

In [Lembo *et al.*, 2002; Calì *et al.*, 2003b; Calì *et al.*, 2002a; Calì *et al.*, 2002c], the authors deal with CQA in GAV integration systems. In their approach a different semantics is used, where insertions are preferred over deletions when repairing the retrieved database. Another difference, is that they consider only inclusions and functional dependencies with some restrictions over the interactions between them.

6.6   Conclusions

We have presented a general approach to specifying, by means of disjunctive logic programs with stable model semantics, the database repairs of a virtual data integration system with open sources under the LAV and GAV approaches.

Consistent answers to queries posed to such a system are computed by running a query program together with the specification of database repairs under the skeptical

or cautious stable model semantics.

The specification of the repairs is achieved by first specifying the class of minimal global legal instances of the integration system (without considering any global ICs at this level yet). To the best of our knowledge, this is also the first specification, under the LAV paradigm, of such global instances in a logic programming formalism. The specification is inspired by the inverse rules algorithms, where auxiliary functions are replaced by auxiliary predicates that are forced to be functional by means of the non deterministic choice operator.

The specification of the minimal legal instances of the integration system allows obtaining the *minimal answers* to arbitrary queries; and the *certain answers* to monotone queries, what extends previous results in the literature related to query plan generation under the LAV approach.

The methodology for specifying minimal legal instances, computing certain answers, and CQA works for conjunctive view definitions and disjunctions thereof. Wrt the ICs and queries this approach can handle, the solution is sound and complete for combinations of universal ICs and acyclic referential ICs, and queries expressed as Datalog$^{\neg}$ programs. In consequence, the current approach to consistent query answering (CQA) subsumes and extends the methodologies presented in [Bertossi *et al.*, 2002] for integration systems, and the one in [Barceló *et al.*, 2003] for stand alone relational databases. Also the complexity of query evaluation using the logic programs presented here matches the theoretical lower bounds for computing certain and consistent answers.

The specifications of the retrieved and minimal databases given for the GAV and LAV approaches can be easily combine to provide a specification for the GLAV approach [Friedman *et al.*, 1999]. By combining this specification with the repair program, we would also be able to give consistent answers for DIS under the GLAV

mapping.

We have already indicated that, in the case the set of ICs contain referential ICs with cycles, the stable models of the specification programs may correspond to a superclass of the repairs of the global system [Bravo and Bertossi, 2006].

Most of the results in this chapter have been published in [Bravo and Bertossi, 2003; Bravo and Bertossi, 2005; Bertossi and Bravo, 2005].

Wrt related work, query answering in virtual DISs *under the assumption* that certain global ICs hold has been treated in [Gryz, 1999; Duschka *et al.*, 2000; Grant and Minker, 2002; Calì *et al.*, 2002a]. However, in CQA we do not assume that global ICs hold. Logic programming specifications of repairs of single relational databases have been presented in [Arenas *et al.*, 2003; Greco *et al.*, 2001; Barceló; and Bertossi, 2003].

In [Bertossi *et al.*, 2002], CQA in possibly inconsistent integration systems under the LAV approach is considered. There, the notion of repair of a minimal legal instance is introduced. The algorithm for CQA is based on a query transformation mechanism [Arenas *et al.*, 1999] applied to first-order queries. The resulting query may contain negation, and is run on top of an extension of the inverse algorithm to the case of stratified Datalog^{not} queries. This approach is limited by the restrictions of the query transformation methodology [Arenas *et al.*, 1999]. In particular, it can be applied only to queries that are quantifier-free conjunctions of literals, and to universal ICs.

Integration systems under the GAV approach that do not satisfy global key dependencies are considered in [Lembo *et al.*, 2002]. There, legal instances are allowed to be more flexible, allowing their computed views to accommodate the satisfaction of the ICs. In this sense, the notion of repair is implicit; and the legal instances are the repairs we have considered here. View definitions are expressed as Datalog queries;

and the queries to the global system are conjunctive. The "repairs" of the global system are specified by normal programs under stable model semantics. In [Calì *et al.*, 2003b], and still under the GAV approach, this work is extended by introducing rewriting techniques to retrieve the consistent query answers without constructing the "repairs". More related work is discussed in the survey [Bertossi and Bravo, 2005] .

In this chapter, we have considered repairs based on null values for RICs. In [Arenas *et al.*, 2003; Barceló; and Bertossi, 2003; Calì *et al.*, 2003a], repairs of RICs using non-null domain values are considered. Under cyclic sets of RICs, this may lead to undecidability of consistent query answering.

Research related to the design of virtual data integration systems and its impact on global query answering has been mostly neglected. Most of the research in the area starts from a given set of view definitions, but the conditions on them hardly go beyond classifying them as conjunctive, disjunctive, Datalog, etc. However, other conditions, imposed when the systems is being designed, could have an impact on, e.g. query plan derivation. Much research is needed in this direction.

Chapter 7

Consistency in Peer Data Management Systems

In order to answer a query, a peer P may need to consider both its own data and the data stored at other peers' sites if the latter are related to P by *Data Exchange Constraints* (DECs). Keeping P's exchange constraints satisfied may imply not only getting data from other peers to complement its own data, but also not using part of its own data. Moreover, the decision does not depend only on the exchange constraints, but also on the *trust relationships* that P has with other peers. For example, if peer P trusts peer Q's data more than its own, P will accommodate its data to Q's in order to keep the exchange constraints satisfied. Another element to take into account in this process is a possible set of local semantic constraints that each individual peer may have. The peer will also need to repair its own data to return consistent answers.

Given a network of peers, each with its own data, and a particular peer P in it, a *solution for* P is -loosely speaking- a global database instance that respects the exchange constraints and trust relationships P has with its immediate neighbors and stays as close as possible to the available data in the system. Since the answers from P have to be consistent with respect to both the local semantic constraints and the data exchange constraints with other peers, the *peer consistent answers* (PCAs) from P are defined as those answers that can be retrieved from P's portion of data in *every* possible solution for P. This definition may suggest that P may change other peers' data, specially of those it considers less reliable, but this is not the case. The notion of solution is used as an auxiliary notion to characterize the correct

answers from P's point of view. Ideally, P should be able to obtain its peer consistent answers just by querying the already available local instances. This resembles the approach to *consistent query answering* (CQA) in databases [Arenas *et al.*, 1999; Bertossi and Chomicki, 2003].

In this chapter, we give a precise semantics for peer consistent answers to first-order queries. First for the *direct case*, where transitive relationships between peers via DECs are not automatically considered; and secondly, the *transitive case*. For the transitive case we provide three possible semantics that consider different granularity of data that can be sent between the different peers, and also provide mechanisms for obtaining PCAs. The semantics are based on a specification of the solutions for a peer as the stable models of a logic program, which captures the different ways the system stabilizes after satisfying the DECs and the trust relationships.

7.1   A Framework for P2P Data Exchange

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

Definition 7.1 A *P2P data exchange system* $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, \textit{trust} \rangle$ consists of:

1. A finite set $\mathcal{P}$ of peers, denoted by A, B, C, ..., P, Q, ...

2. For each peer P, a database schema $\mathcal{R}(\mathtt{P})$ that includes a domain $\mathcal{U}(\mathtt{P})$, and relations $R(\mathtt{P}), \dots$. However, it may be convenient to assume that all peers share a common, fixed and possibly infinite domain $\mathcal{U}$. Each $\mathcal{R}(\mathtt{P})$ determines a first-order (FO) language $\mathcal{L}(\mathtt{P})$. We assume that the schemas $\mathcal{R}(\mathtt{P})$ are disjoint except for the elements in their domains. $\mathcal{R}$ denotes the union of the $\mathcal{R}(\mathtt{P})$s.

3. For each peer P, a database instance $D(\mathtt{P})$ corresponding to schema $\mathcal{R}(\mathtt{P})$.

4. For each peer P, a set of $\mathcal{L}(\text{P})$-sentences $IC(\text{P})$ that are ICs on $\mathcal{R}(\text{P})$. Let $IC = \bigcup_{\text{P}} IC(\text{P})$.

5. For each peer P, a collection $\Sigma(\text{P})$ of *sets of data exchange constraints* $\Sigma(\text{P}, \text{Q})$ consisting of sentences written in the FO language for the signature $\mathcal{R}(\text{P}) \cup \mathcal{R}(\text{Q})$, and the Q's are (some of the) other peers in $\mathcal{P}$. Let $\Sigma = \bigcup_{\text{P} \in \mathcal{P}} \Sigma(\text{P})$.

6. A relation $trust \subseteq \mathcal{P} \times \{less, same\} \times \mathcal{P}$, with the intended semantics that when $(\text{A}, less, \text{B}) \in trust$, peer A trusts itself less than B; while $(\text{A}, same, \text{B}) \in trust$ indicates that A trusts itself the same as B. In this relation, the second argument functionally depends on the other two. By default, a peer trusts its own data more than that of others. □

Each peer P is responsible for maintaining its material instance consistent with respect to $IC(\text{P})$, independently from other peers. However, when local data is virtually changed to accommodate to other peers' data, the local ICs could be virtually violated. It is possible to keep the local ICs satisfied also at query time by using methodologies developed for consistent query answering [Bertossi and Chomicki, 2003]. A peer may submit queries to other peers in accordance with the restrictions imposed by its DECs and using other peers' relations appearing in them.

Definition 7.2 (a) Give a set of peers $\mathcal{P}$, we denote with $\overline{\mathcal{R}}(\mathcal{P})$ the schema consisting of the union of $\mathcal{R}(\text{P})$ for all $\text{P} \in \mathcal{P}$. (b) We use $\bar{D}(\mathcal{P})$ to denote the database instance on $\overline{\mathcal{R}}(\mathcal{P})$, consisting of the union of the $D(\text{P})$s for all $\text{P} \in \mathcal{P}$. (c) If D is an instance for some schema $\mathcal{S}$ and $\mathcal{S}'$ is a subschema of $\mathcal{S}$, then $D|\mathcal{S}'$ denotes the restriction of D to $\mathcal{S}'$. In particular, if $\mathcal{R}(\text{P}) \subseteq \mathcal{S}$, then $D|\text{P}$ denotes the restriction of D to $\mathcal{R}(\text{P})$. (d) We denote by $\mathcal{R}(\text{P})^{less}$ the union of all schemas $\mathcal{R}(\text{Q})$, with $(\text{P}, less, \text{Q}) \in trust$. $\mathcal{R}(\text{P})^{same}$ is analogously defined. □

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

The semantics of peer consistent query answers for a peer P is given in terms of all possible minimal, virtual and simultaneous repairs of the local databases that lead to a satisfaction of the DECs while respecting P's trust relationships to other peers. This repair process may lead to alternative global databases called the *global solutions* for P. The set of *solutions* for P are the global solutions restricted to the relations in P. Next, the peer consistent answers from P are those that are invariant with respect to all its solutions.

Definition 7.3 The *accessibility graph* $\mathcal{G}_A(\mathfrak{P})$ for a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$ is defined as follows: Each peer $\text{P} \in \mathcal{P}$ is a vertex, and there is a directed edge from Pi to Pj iff there exists a DEC in $\Sigma(\text{Pi}, \text{Pj})$ and $(\text{Pi}, less, \text{Pj}) \in trust$ or $(\text{Pi}, same, \text{Pj}) \in trust$. The directed edges are labelled with "<" or "=". For $\mathcal{S} \subseteq \mathcal{P}$, $\mathcal{G}_A(\mathfrak{P})[\mathcal{S}]$ is the restriction of $\mathcal{G}_A(\mathfrak{P})$ to the vertices in $\mathcal{S}$. □

If there is an edge from Pi to Pj labelled with "<", this means that Pi trusts itself less than Pj. If it is labelled with "=", it means Pi trusts itself as much as it trusts Pj. Note that we are not interested in a DEC in $\Sigma(\text{P1}, \text{P2})$ where peer P1 trusts itself more than peer P2, since that means that the information of P2 is not relevant to P1.

Example 7.1 Consider the P2P data exchange system $\mathfrak{P}$:

P1: $\mathcal{R}(\texttt{P1}) = \{R^1(\cdot, \cdot)\}$, $D(\texttt{P1}) = \{R^1(a, b), R^1(s, t)\}$,

P2: $\mathcal{R}(\texttt{P2}) = \{R^2(\cdot, \cdot)\}$, $D(\texttt{P2}) = \{R^2(c, d), R^2(a, e)\}$,

P3: $\mathcal{R}(\texttt{P3}) = \{R^3(\cdot, \cdot)\}$, $D(\texttt{P3}) = \{R^3(s, u)\}$,

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

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

The trust relationships are shown in the accessibility graph in Figure 7.1. For example, P1 trusts itself less than it trusts peer P2, therefore there is an edge from P1 to P2 labeled with $<$. On the other hand, P3 has no trust relationship, so there is no edge starting from it. $\qquad \square$
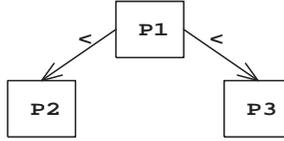
Figure 7.1: Accessibility graph of Example 7.1

Definition 7.4 A peer P' is *accessible* from peer P if there is a path in the directed graph $\mathcal{G}_A(\mathfrak{P})$ from P to P' or if P'=P. A peer P' is a *neighbor* of P if there is an edge from P to P' in $\mathcal{G}_A(\mathfrak{P})$, or if P' = P. Let $\mathcal{AC}(\texttt{P})$ and $\mathcal{N}(\texttt{P})$ be the set of peers that are accessible from P and the neighbors of P, respectively. $\qquad \square$

Example 7.2 Consider the following P2P data exchange system $\mathfrak{P}$ with four peers with their DECs:

$\mathcal{R}(\texttt{P1}) = \{R^1(\cdot, \cdot)\}$, $\qquad \mathcal{R}(\texttt{P2}) = \{R^2(\cdot, \cdot), S^2(\cdot, \cdot)\}$,

$\mathcal{R}(\texttt{P3}) = \{R^3(\cdot, \cdot)\}$, $\qquad \mathcal{R}(\texttt{P4}) = \{R^4(\cdot, \cdot, \cdot)\}$,

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

$\Sigma(\texttt{P2}, \texttt{P3}) = \{\forall x \forall y (R^2(x, y) \wedge R^3(x, y) \rightarrow \textbf{false})\}$,

$$\Sigma(\texttt{P4},\texttt{P2})= \{\forall x\forall y\forall z(R^2(x,y)\wedge S^2(y,z)\rightarrow R^4(x,y,z))\},$$

$$\Sigma(\texttt{P4},\texttt{P3})= \{\forall x\forall y\forall z(R^4(x,y,z)\rightarrow R^3(x,z))\},$$

$$trust = \{(\texttt{P1},\textit{less},\texttt{P2}),\ (\texttt{P2},\textit{same},\texttt{P3}),\ (\texttt{P4},\textit{less},\texttt{P2}),\ (\texttt{P4},\textit{less},\texttt{P3})\}.$$

The accessible peers for each peer are the following: $\mathcal{AC}(\texttt{P1}) = \{\texttt{P1},\texttt{P2},\texttt{P3}\}$, $\mathcal{AC}(\texttt{P2}) = \{\texttt{P2},\texttt{P3}\}$, $\mathcal{AC}(\texttt{P3}) = \{\texttt{P3}\}$ and $\mathcal{AC}(\texttt{P4}) = \{\texttt{P2},\texttt{P3},\texttt{P4}\}$. Figure 7.2 shows the accessibility graph $\mathcal{G}_A(\mathfrak{P})$, $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\texttt{P1})]$ and $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\texttt{P4})]$.

(a) $\mathcal{G}_A(\mathfrak{P})$ (b) $\mathcal{G}_A(\mathfrak{P})$ restricted to $\mathcal{AC}(\texttt{P1})$ (c) $\mathcal{G}_A(\mathfrak{P})$ restricted to $\mathcal{AC}(\texttt{P4})$
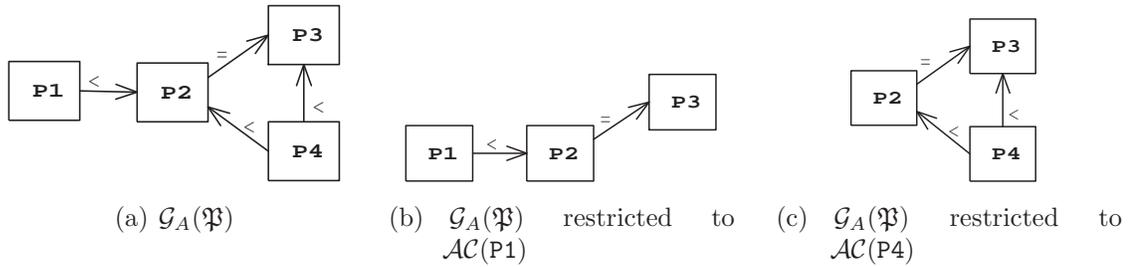
Figure 7.2: Accessibility graphs of Example 7.2

The neighborhoods are the following: $\mathcal{N}(\texttt{P1}) = \{\texttt{P1},\texttt{P2}\}$, $\mathcal{N}(\texttt{P2}) = \{\texttt{P2},\texttt{P3}\}$, $\mathcal{N}(\texttt{P3}) = \{\texttt{P3}\}$, and $\mathcal{N}(\texttt{P4}) = \{\texttt{P2},\texttt{P3},\texttt{P4}\}$. $\qquad\square$

7.2   The Direct Case

A peer's solution captures the idea that only some peers' databases are relevant to P. In the direct case, we will assume that the only data and DECs that are relevant are the ones of the neighbors of P. In this sense, this is a "local notion", because it does not take into consideration transitive dependencies. In Section 7.3 we will address the transitive case.

Definition 7.5 (direct case) Given a peer P in a P2P data exchange system and an instance D on $\overline{\mathcal{R}}(\mathcal{N}(\texttt{P}))$, an instance D' on $\overline{\mathcal{R}}(\mathcal{N}(\texttt{P}))$ is a *neighborhood solution for* P if D' is a repair of D with respect to $\Sigma(\texttt{P})\cup IC(\texttt{P})$ that does not change the more trusted relations, more precisely: (a) $D' \models \bigcup\{\Sigma(\texttt{P},\texttt{Q})\mid$ where P and Q are adjacent

in $\mathcal{G}_A(\mathfrak{P})\} \cup IC(\texttt{P})$; (b) $D'|P = D|P$ for every predicate $P \in \mathcal{R}(\texttt{Q})$, where there is an edge from $\texttt{P}$ to $\texttt{Q}$ labelled $<$; (c) D' minimally differs from D in the sense that $(D' \smallsetminus D) \cup (D \smallsetminus D')$ is minimal under set inclusion among those instances that satisfy (a) and (b).

An instance D is a *local solution for* peer $\texttt{P}$ if there is a neighborhood solution D' for $\texttt{P}$ such that $D = D'|\mathcal{R}(P)$. □

Intuitively, a neighborhood solution for $\texttt{P}$ repairs the instance of the peer and its neighbors with respect to the DECs with peers that $\texttt{P}$ trusts more than –or the same as– itself, but leaving unchanged the tables that belong to more trusted peers. As a consequence of the definition, tables belonging to peers that are not related to $\texttt{P}$ or are less trustable are not changed. In other words, $\texttt{P}$ tries to change its own tables according to what the dependencies to more or equally trusted peers prescribe.

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

In these definitions we find clear similarities with the characterization of consistent query answers in single relational databases [Bertossi and Chomicki, 2003]. However, in P2P query answering, repairs may involve data associated to different peers, and also a notion of priority that is related to the trust relation.

Example 7.3 (example 7.1 continued) Since $\texttt{P1}$ trusts $\texttt{P2}$ and $\texttt{P3}$ more than itself, $\texttt{P2}$ can solve the inconsistencies with respect to the DECs at query time by virtually modifying its own data. For $\Sigma(\texttt{P1},\texttt{P2})$, we have $(c,d),(a,e) \in R^2$, but $(c,d),(a,e) \notin$

R^1, then, in order to satisfy the DEC, P1 should virtually add those tuples to R^1. $\Sigma(\text{P1}, \text{P3})$ is violated by $R^3(s, u)$ and $R^1(s, t)$. Since P1 trusts itself less than P3, peer P1 will ignore tuple $R^1(s, t)$ in order to restore consistency. The unique neighborhood solution for peer P1 according to Definition 7.5 is $\{R^3(s, u), R^2(a, e), R^2(c, d), R^1(a, e),$ $R^1(c, d), R^1(a, b)\}$ and the respective local solution is $\{R^1(a, e), R^1(c, d), R^1(a, b)\}$. □

Example 7.4 Consider a P2P data exchange system:

P1: $\mathcal{R}(\text{P1}) = \{R^1(\cdot, \cdot)\}$, $D(\text{P1}) = \{R^1(a, b), R^1(s, t)\}$,

P2: $\mathcal{R}(\text{P2}) = \{R^2(\cdot, \cdot)\}$, $D(\text{P2}) = \{R^2(c, d), R^2(a, e)\}$,

P3: $\mathcal{R}(\text{P3}) = \{R^3(\cdot, \cdot)\}$, $D(\text{P3}) = \{R^3(a, f), R^3(s, u)\}$,

$\Sigma(\text{P1}, \text{P2}) = \{\forall xy(R^2(x, y) \rightarrow R^1(x, y))\}$,

$\Sigma(\text{P1}, \text{P3}) = \{\ \forall xyz(R^1(x, y) \wedge R^3(x, z) \rightarrow y = z)\ \}$,

$trust = \{\ (\text{P1}, less, \text{P2}),\ (\text{P1}, same, \text{P3})\ \}$.

In this case, $\bar{D}(\mathcal{N}(\text{P1})) = \{R^1(a, b),\ R^1(s, t), R^2(c, d), R^2(a, e), R^3(a, f), R^3(s, u)\}$. It has two neighborhood solutions according to Definition 7.5, namely $D' = \{R^1(a, b),$ $R^1(s, t), R^1(c, d), R^1(a, e), R^2(c, d), R^2(a, e)\}$; and $D'' = \{\ R^1(a, b), R^1(c, d), R^1(a, e),$ $R^2(c, d), R^2(a, e), R^3(s, u)\}$. Therefore, the local solutions for peer P1 are $D'|\mathcal{R}(\text{P1}) =$ $\{R^1(a, b), R^1(s, t), R^1(c, d), R^1(a, e)\}$ and $D''|\mathcal{R}(\text{P1}) = \{\ R^1(a, b), R^1(c, d), R^1(a, e)\}$. □

Definition 7.6 Given a FO query $Q(\bar{x}) \in \mathcal{L}(\text{P})$ posed to P, a ground tuple $\bar{t}$ is a *peer consistent* answer to Q for P iff $D' \models Q(\bar{t})$ for every local solution D' for P. □

Example 7.5 (example 7.4 continued) The query $Q : R^1(x, y)$ posed to P1 has as peer consistent answers (PCA) the tuples $(a, b), (c, d), (a, e)$, because those are the tuples found in all local solutions for peer P1 . □

Notice that this definition of PCA is relative to a fixed peer, and not only because the query is posed to one peer and in its query language, but also because this notion is based on the "direct or local" notion of a solution for a single peer, which considers its "direct neighbors" only. This is a first step towards the general case of transitive dependencies that will be explored in Section 7.3. However, this restricted case is the basis for the transitive case, because P does not see beyond its neighbors. When P requests data to a neighbor, say Q, the latter may have to find local solutions of its own by considering its direct neighbors. The transitive case has to combine these local solutions (see Section 7.3).

Peer consistent answers to queries can be obtained by using techniques similar to those of CQA, e.g. query rewriting [Arenas *et al.*, 1999; Bertossi and Chomicki, 2003]. However, there are important differences, because there are now some fixed predicates in the repair process.

Example 7.6 (example 7.4 continued) If P1 is posed the query $Q: R^1(x, y)$, asking for the tuples in relation R^1, its answers can be obtained through the rewritten query $Q' : [R^1(x, y) \wedge \forall z_1((R^3(x, z_1) \wedge \neg \exists z_2 R^2(x, z_2)) \rightarrow z_1 = y)] \quad \vee \quad R^2(x, y)$, which requires from P1 to submit queries to its neighboring peers. Peer P1 needs to query peer P2 to get all the tuples in R^2. Peer P3 could be queried once to get all the tuples in R^3 and then apply the query locally in P1, or it could receive several queries to get the tuples in R^3 such that the first attribute is in the first attribute of R^1 and not in R^2. The final answers are $(a, b), (c, d)$ and (a, e), precisely the answers obtained in Example 7.5. □

Notice that a query Q may have peer consistent answers for a peer which are not answers to Q when the peer is considered in isolation; this is because the peer may

import data from other peers.[1]

This query rewriting approach differs from the one used for CQA. In the latter case, literals in a query are resolved (by *resolution*) against ICs in order to generate residues that are iteratively appended as extra conditions to the query [Arenas *et al.*, 1999]. In the case of P2P data exchange systems, the query may have to be modified in order to include new data that is located at a different peer's site. This cannot be achieved by imposing extra conditions alone, but instead, by relaxing the query in some sense. Since query answering in P2P data exchange systems includes sufficiently complex cases of CQA, a FO query rewriting approach to P2P query answering is bound to have limitations in terms of completeness [Bertossi and Chomicki, 2003]. Instead, we will now propose a more general methodology based on answer set programming [Baral, 2003; Gelfond and Leone, 2002].

7.2.1 The Solution Program

A logic programming approach to the specification of solutions for a peer can be developed. Those specifications will be similar to those of repairs of single relational databases under referential integrity constraints [Barceló *et al.*, 2003]. However, as we have seen, there are important differences with CQA.

We now give an example of an involved exchange constraint that shows the main issues around this kind of specifications.

Example 7.7 Consider a P2P data exchange system:

P1: $\mathcal{R}(\text{P1}) = \{R^1(\cdot, \cdot), S^1(\cdot, \cdot)\}$,

P2: $\mathcal{R}(\text{P2}) = \{R^2(\cdot, \cdot), S^2(\cdot, \cdot)\}$,

$\Sigma(\text{P1}, \text{P2}) = \{ \forall x \forall y \forall z \exists w (R^1(x, y) \wedge R^2(z, y) \rightarrow S^1(x, w) \wedge S^2(z, w)) \}$,

---

[1] Another difference with CQA, where all consistent answers are answers to the original query; at least for conjunctive queries and *generic* ICs [Bertossi and Chomicki, 2003].

$trust = \{(\texttt{P1}, less, \texttt{P2})\}.$

The DEC mixes tables of the two peers on each side of the implication. If $\Sigma(\texttt{P1}, \texttt{P2})$ is satisfied by the combination of the data in P1 and P2, then the current global instance constitutes P1's solution. Otherwise, alternative solutions for P1 have to be found, keeping P2's data fixed in the process. This is the case when there are ground tuples $R^1(d, m) \in D(\texttt{P1})$, $R^2(a, m) \in D(\texttt{P2})$, such that for no t it holds both that $S^1(d, t) \in D(\texttt{P1})$ and $S^2(a, t) \in D(\texttt{P2})$.

Obtaining peer consistent answers for peer P1 amounts to virtually restoring the satisfaction of $\Sigma(\texttt{P1}, \texttt{P2})$ by virtually modifying P1's data. In order to specify P1's modified relations, we introduce virtual versions $R^{1'}, S^{1'}$ of R^1, S^1, containing the data in peer P1's solutions. In consequence, at the solution level, we have the relations $R^{1'}, S^{1'}, R^2, S^2$. Since P1 is querying its database, its original queries will be expressed in terms of relations $R^{1'}, S^{1'}$ only (plus, possibly, built-ins).

The contents of the virtual relations[2] R'_1, R'_2 are obtained from the material sources R^1, S^1, R^2, S^2. Since R^2, S^2 are fixed, the satisfaction of $\Sigma(\texttt{P1}, \texttt{P2})$ requires $R^{1'}$ to be a subset of R^1, and $S^{1'}$, a superset of S^1. The specification of these relations is done in extended disjunctive logic programs with answer set (stable model) semantics [Gelfond and Lifschitz, 1991]. These programs add classical negation to disjunctive logic programs and their semantics is given by answer sets, which are a generalization of the stable models. This type of programs are also used in [Arenas *et al.*, 2003] to specify database repairs.

The first rules for the specification program Π are:

$$R^{1'}(x, y) \leftarrow R^1(x, y), \; not \; \neg R^{1'}(x, y). \qquad S^{1'}(x, y) \leftarrow S^1(x, y), \; not \; \neg S^{1'}(x, y)., \quad (7.1)$$

---

[2]We can observe that the virtual relations can be seen as virtual global relations in a virtual data integration system [Bertossi and Bravo, 2004b; Levy, 2000; Lenzerini, 2002].

which specify that, by default, the tuples in the source relations are copied into the new virtual versions, but with the exception of those that may have to be removed in order to satisfy $\Sigma(\texttt{P1}, \texttt{P2})$ (with R^1, S^1 replaced by $R^{1'}, S^{1'}$). Some of the exceptions for R_1' are specified by

$$\neg R^{1'}(x, y) \leftarrow R^1(x, y), R^2(z, y), \; not \; aux_1(x, z), \; not \; aux_2(z). \tag{7.2}$$

$$aux_1(x, z) \leftarrow S^1(x, w), S^2(z, w). \qquad aux_2(z) \leftarrow S^2(z, w). \tag{7.3}$$

That is, $R^1(x, y)$ is deleted if it participates in a violation of $\Sigma(\texttt{P1}, \texttt{P2})$ (what is captured by the first three literals in the body of (7.2) plus the first rule in (7.3)), and there is no way to restore consistency by inserting a tuple into S^1, because there is no possible matching tuple in S^2 for the possibly new tuple in S^1 (what is captured by the last literal in the body of (7.2) plus the second rule in (7.3)). In case there is such a tuple in S^2, we either delete a tuple from R^1 or insert a tuple into S^1:

$$\neg R^{1'}(x, y) \vee S^{1'}(x, w) \quad \leftarrow \quad R^1(x, y), R^2(z, y), \; not \; aux_1(x, z), S^2(z, w),$$

$$choice((x, z), w). \tag{7.4}$$

That is, in case of a violation of $\Sigma(\texttt{P1}, \texttt{P2})$, when there is tuple of the form (a, t) in S^2 for the combination of values (d, a), then the *choice operator* [Giannotti *et al.*, 1991] non deterministically chooses a unique value for t, so that the tuple (d, t) is inserted into S^1 as an alternative to deleting (d, m) from R^1. The *choice* predicate can be replaced by a standard predicate plus extra rules that choose a unique value for t [Giannotti *et al.*, 1991]. No exceptions are specified for $S^{1'}$, which makes sense since $S^{1'}$ is a superset of S^1. Then, the negative literal in the body of (7.1) can be eliminated. However, new tuples can be inserted into $S^{1'}$, which is captured by rule (7.4). Finally, the program must contain the tuples in the relations R^1, S^1, R^2, S^2 as

facts.

If `P1` equally trusts itself and `P2`, both `P1` and `P2`s' relations are flexible when searching for a solution. Thus, the program becomes more involved, because now R^2, S^2 may also change; and virtual versions for them must be specified.　　　□

This example shows the main issues in the specification of a peer's solutions. The program with choice operators can be translated into one with standard answer set (or stable model) semantics [Giannotti *et al.*, 1991]; and the solutions are in one-to-one correspondence with the answer sets of the program. Actually, each answer set S corresponds to a neighborhood solution $D'(S)$ for peer `P` which coincides with the original material global instance on the tables other than R_1, R_2, whereas for the latter, the contents are of the form $\{\bar{t} \mid R_i'(\bar{t}) \in S\}, i = 1, 2$, resp. The absence of solutions for a peer is captured through the non-existence of answer sets for program Π.

Since program Π represents all the solutions for a peer in a compact form, the peer consistent answers from a peer can be obtained by running a query program expressed in terms of the virtually repaired tables in combination with the specification program Π. For this, the combined program is run under the skeptical answer set semantics, for which a system like DLV [Leone *et al.*, 2006] can be used. For example, the query $Q(x, z) : \exists y(R_1(x, y) \wedge R_2(z, y))$ issued to peer `P`, would be peer–consistently answered by running the query program $Ans_Q(x, z) \leftarrow R_1'(x, y), R_2'(x, y)$ together with program Π. Although only (the new versions of) `P`'s relations appear in the query, the program may make `P` import `Q`'s data.

If a peer `P` has local ICs $IC(\texttt{P})$ to be satisfied, also at query time, then the program that specifies its solutions should take care of its ICs. A simple but radical way of doing this consists of using program denial constraints. If in Section 7.2.1 we had, for peer `P`, the local functional dependency (FD) $\forall x \forall y \forall z(R_1(x, y) \wedge$

$R_1(x, z) \rightarrow y = z$), then the program would include the program denial constraint $\leftarrow R_1(x, y), R_1(x, z), y \neq z$, having the effect of pruning those solutions that do not satisfy the FD. However, a more flexible –or "robust" [Franconi *et al.*, 2004b]– alternative for keeping the local ICs satisfied consists in having the specification program split in two layers, where the first one builds the solutions, without considering the local ICs, and where the second one repairs the solutions with respect to the local ICs, as done with single inconsistent relational databases [Barceló *et al.*, 2003].

A more uniform approach consists in identifying $IC(\mathsf{P})$ with $\Sigma(\mathsf{P}, \mathsf{P})$ and considering $(\mathsf{P}, same, \mathsf{P}) \in trust$. The notion of solution for a peer given in Definition 7.5 captures this idea.

For DECs with existential quantifiers, it is necessary to choose values from a domain. There are several options, some of them already considered for CQA: (a) take a value from an open infinite domain; (b) assign labelled null values [Fagin *et al.*, 2005]; (c) take a value from an appropriate finite and closed proper superset of the active domains [Bravo and Bertossi, 2005]; (d) introduce fresh constants, whenever needed, from a separate domain [Calvanese *et al.*, 2004a]. The option taken and the class of DECs (e.g. presence cycles) may determine, e.g. decidability of peer consistent answering [Chomicki and Marcinkowski, 2002; Chomicki and Marcinkowski, 2005b; Calì *et al.*, 2003a; Halevy *et al.*, 2003; Calvanese *et al.*, 2004a; Bravo and Bertossi, 2006].

If the DEC has a disjunction of literals in the consequent, it is possible to replace the choice operator and replace the existentially quantified variables by *null*, i.e., a non-labelled null value [Barceló *et al.*, 2003; Bravo and Bertossi, 2006]. This cannot be done in the case of Example 7.7 since there would not be a join if we replace the two occurrences of w by *null* (see Chapter 5). In what follows, we will restrict to DECs where the consequent is a disjunction of literals. Note, that these DECs might

still combine predicates of different peers in the antecedent and consequent.

Definition 7.7 A *universal data exchange constraint* (UDEC) in $\Sigma(\texttt{P1},\texttt{P2})$ is a DEC of form:

$$\forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \longrightarrow (\bigvee_{j=1}^{m} Q_j(\bar{y}_j) \vee \varphi)). \tag{7.5}$$

where, for $R = \{R_i \mid i \in \{1,\ldots,n\}\}$ and $Q = \{Q_j \mid j \in \{1,\ldots,m\}\}$, $R \cup Q \subseteq \overline{\mathcal{R}}(\{\texttt{P1},\texttt{P2}\})$, $(R \cup Q) \cap \mathcal{R}(\texttt{P1}) \neq \emptyset$, $(R \cup Q) \cap \mathcal{R}(\texttt{P2}) \neq \emptyset$, .

A *referential data exchange constraint* (RDEC) in $\Sigma(\texttt{P1},\texttt{P2})$ is a DEC of form:

$$\forall \bar{x}(R(\bar{x}) \longrightarrow \exists \bar{y}\ Q(\bar{x}',\bar{y})), \tag{7.6}$$

where $R, Q \subseteq \overline{\mathcal{R}}(\{\texttt{P1},\texttt{P2}\})$, $\{R,Q\} \cap \mathcal{R}(\texttt{P1}) \neq \emptyset$, and $\{R,Q\} \cap \mathcal{R}(\texttt{P2}) \neq \emptyset$. $\quad\square$

A wide class of DECs can be accommodated into equations (7.5) and (7.6). However, tuple-generating-dependencies (TGDs) [Beeri and Vardi, 1984] cannot. TGDs are used in the context of data exchange [Fagin *et al.*, 2003a; Fagin *et al.*, 2003b; Kolaitis *et al.*, 2006] and are of the form:

$$\forall \bar{x}(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \longrightarrow \bigwedge_{j=1}^{m} Q_j(\bar{y}_j)). \tag{7.7}$$

We do not consider this type of constraints, since we want to repair inconsistencies with respect to DECs using *null*, as done in Chapters 5 and 6 in the context of databases and data integration systems. Inconsistencies with respect to TGDs cannot be enforced by the use of *null* since a joint in the consequent can not be satisfied by it.

Example 7.8 The DECs in Example 7.2 are universal. The DEC in Example 7.7 is a TGD but not a UDEC nor a RDEC. An example of a RDEC is: $\Sigma(\texttt{P},\texttt{Q}) =$

$$\forall x(T^1(x, y) \rightarrow \exists z T^2(x, z)). \qquad \qquad \qquad \Box$$

In a PDM we can have two sources for cycles: trust relationships and constraints (DECs and ICs). There is a *cycle through trust relationships* if the graph $\mathcal{G}_A(\mathfrak{P})$ is cyclic. There is a *cycle through constraints* if $\Sigma \cup IC$ is RIC-cyclic[3] (see Definition 2.2). Note that if there is no cycle through trust relationships, the only way for the PDM to have a cycle through constraints is if a peer's ICs are RIC-cyclic.

Definition 7.5 corresponds to repairing the DECs by using the choice operator over an infinite domain. Now, since we are considering only UDECs and RDECs, we can repair instead by using *null*:

Definition 7.8 (direct case with null values) Given a peer P in a P2P data exchange system and an instance D on $\overline{\mathcal{R}}(\mathcal{N}(\mathtt{P}))$, an instance D' on $\overline{\mathcal{R}}(\mathcal{N}(\mathtt{P}))$ is a *neighborhood solution for* P if D' is a repair of D with respect to $\Sigma(\mathtt{P}) \cup IC(\mathtt{P})$ that does not change the more trusted relations, more precisely: (a) $D' \models \bigcup\{\Sigma(\mathtt{P}, \mathtt{Q}) \mid (\mathtt{P}, less, \mathtt{Q})$ or $(\mathtt{P}, same, \mathtt{Q}) \in trust\} \cup IC(\mathtt{P})$; (b) $D'|P = D|P$ for every predicate $P \in \mathcal{R}(\mathtt{Q})$, where Q is a peer with $(\mathtt{P}, less, \mathtt{Q}) \in trust$; (c) There does not exist an instance D'' that satisfies (a) and (b), and such that $D'' <_D D'$, where $<_D$ is defined as in Definition 5.3.

An instance D is a *local solution for* peer P if there is a neighborhood solution D' for P such that $D = D'|\mathcal{R}(P)$. $\qquad \qquad \Box$

Inspired by the repair programs for stand alone databases, the following logic program is a specification of the neighborhood solutions for the solution semantics of Definition 7.8.

Definition 7.9 Consider a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$ and a peer $\mathtt{P} \in \mathcal{P}$. The *direct solution program* $\Pi^{direct}(\mathtt{P}, \mathfrak{P})$ is:

---

[3]The UDECs and RDECs in Σ are treated as UICs and RICs respectively.

1. $dom(x)$, for every $x \in (\mathcal{U} \setminus \{null\})$

2. $R(\bar{a})$, for each atom $R(\bar{a}) \in \bar{D}(\mathcal{N}(\texttt{P}))$.

3. For every UDEC $\psi \in \Sigma(\texttt{P},\texttt{Pj})$ of the form (7.5) such that $\texttt{Pj} \in \mathcal{N}(\texttt{P})$ and there exists $(\texttt{P},\{same \text{ or } less\},\texttt{Pj}) \in trust$, the rules:

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

Here, $\mathcal{A}(\psi)$ is the set of relevant attributes of ψ, and $\bar{\varphi}$ is a conjunction of built-ins that is equivalent to the negation of φ. $R_\texttt{P}$ is defined as follows, for $\mathcal{R} = \{R_i \mid i \in \{1,\ldots,n\}\}$:

$$R_\texttt{P} = \begin{cases} \mathcal{R} \cap \mathcal{R}(\texttt{P}) & \text{if } (\texttt{P}, less, \texttt{P2}) \in trust \\ \mathcal{R} & \text{if } (\texttt{P}, same, \texttt{P2}) \in trust \end{cases}$$

$Q_\texttt{P}$ is defined analogously.

4. For every RDEC $\psi \in \Sigma(\texttt{P},\texttt{Pj})$ of the form (7.6) such that $\texttt{Pj} \in \mathcal{N}(\texttt{P})$ and there exists $(\texttt{P},\{same \text{ or } less\},\texttt{Pj}) \in trust$:

   (a) If $(\texttt{P}, same, \texttt{Pj}) \in trust$, the rules:

   $R_-(\bar{x},\mathbf{f_a}) \vee Q_-(\bar{x}',\overline{null},\mathbf{t_a}) \leftarrow P_-(\bar{x},\mathbf{t^\star}), \ not \ aux_\psi(\bar{x}'), \bar{x}' \neq null.$

   (b) If $(\texttt{P}, less, \texttt{Pj}) \in trust$ and $R \in \mathcal{R}(\texttt{P})$, the rules:

   $R_-(\bar{x},\mathbf{f_a}) \leftarrow P_-(\bar{x},\mathbf{t^\star}), \ not \ aux_\psi(\bar{x}'), \bar{x}' \neq null.$

   (c) If $(\texttt{P}, less, \texttt{Pj}) \in trust$ and $Q \in \mathcal{R}(\texttt{P})$, the rules:

   $Q_-(\bar{x}',\overline{null},\mathbf{t_a}) \leftarrow P_-(\bar{x},\mathbf{t^\star}), \ not \ aux_\psi(\bar{x}'), \bar{x}' \neq null.$

   Plus the auxiliary rules:

   $aux_\psi(\bar{x}') \leftarrow Q(\bar{x}',\overline{null}), \ not \ Q_-(\bar{x}',\overline{null},\mathbf{f_a}), \bar{x}' \neq null,$

   $aux_\psi(\bar{x}') \leftarrow Q_-(\bar{x}',\bar{y},\mathbf{t^\star}), \ not \ Q_-(\bar{x}',\bar{y},\mathbf{f_a}), \bar{x}' \neq null, y_i \neq null, \qquad \text{for every } y_i \in \bar{y}.$

5. For every $\psi \in IC(\texttt{Pi})$ such that ψ is a universal IC of the form (2.2) and $\texttt{Pi} \in \mathcal{N}(\texttt{P})$, the rules:

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

Here, $\mathcal{A}(\psi)$ is the set of relevant attributes for ψ, and $\bar{\varphi}$ is a conjunction of built-ins that is equivalent to the negation of φ.

6. For every $\psi \in IC(\texttt{Pi})$ such that ψ is a referential IC of the form (2.3) and $\texttt{Pi} \in \mathcal{N}(\texttt{P})$, the rules:

$P_{\_}(\bar{x}, \mathbf{f_a}) \vee Q_{\_}(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow P_{\_}(\bar{x}, \mathbf{t}^\star), \; not \;\; aux_\psi(\bar{x}'), \bar{x}' \neq null.$

$aux_\psi(\bar{x}') \leftarrow Q_{\_}(\bar{x}', \overline{null}), \;\; not \; Q_{\_}(\bar{x}', \overline{null}, \mathbf{f_a}), \bar{x}' \neq null.$

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

$aux_\psi(\bar{x}') \leftarrow Q_{\_}(\bar{x}', \bar{y}, \mathbf{t}^\star), \;\; not \; Q_{\_}(\bar{x}', \bar{y}, \mathbf{f_a}), \bar{x}' \neq null, \; y_i \neq null.$

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

$R_{\_}(\bar{x}, \mathbf{f}^\star) \leftarrow dom(\bar{x}), \; not \; R(\bar{x}).$

$R_{\_}(\bar{x}, \mathbf{f}^\star) \leftarrow R_{\_}(\bar{x}, \mathbf{f_a}).$

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

$R_{\_}(\bar{x}, \mathbf{t}^\star) \leftarrow R_{\_}(\bar{x}, \mathbf{t_a}).$

8. For each predicate $R \in \mathcal{R}(\texttt{P})$, the interpretation rule:

$R_{\_}(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow R_{\_}(\bar{x}, \mathbf{t}^\star), \;\; not \; R_{\_}(\bar{x}, \mathbf{f_a}).$

9. For each predicate $R \in \mathcal{R}(\mathcal{N}(\texttt{P}))$, the program denial constraint:

$\leftarrow \; R_{\_}(\bar{x}, \mathbf{t_a}), R_{\_}(\bar{x}, \mathbf{f_a}).$ $\hfill \square$

In rules 3. and 4. of the program, the head of the rules will only include predicates of the less trusted peer, this is, the repair process will only modified the less trusted peers. Rules 5. and 6. treat the ICs of each peer $\texttt{P1} \in \mathcal{N}(P)$ as DECs in $\Sigma(\texttt{P1}, \texttt{P1})$,

with trust relation (P1, *same*, P1). Also, as it can be seen from rules 3., 4., 5. and 6., when adding tuples, *null* is used for unknown values. Atoms labelled with $\mathbf{t}^{\star\star}$ store the database atoms in the neighborhood solution.

Program $\Pi^{direct}(\text{P}, \mathfrak{P})$ differs from the program given in Example 7.7 in two ways. First, inconsistencies with respect to DECs are solved by replacing existential quantifiers by *null* instead of values from the active domain. Second, annotation constants are added to handle interaction between DECs.

Definition 7.10 The P2P instance associated to a stable model $\mathcal{M}$ of program Π^{direct} $(\text{P}, \mathfrak{P})$ is $D_{\mathcal{M}} = \{R(\bar{a}) \mid R(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M} \text{ and } R \in \mathcal{R}(\mathcal{N}(\text{P}))\}$. $\qquad \square$

Proposition 7.1 Given a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$, such that $\Sigma \cup IC$ is RIC-acyclic,[4] and a peer P in $\mathcal{P}$, a database D' is a neighborhood solution with *null* (see Definition 7.8) iff there exist a stable model $\mathcal{M}$ of $\Pi^{direct}(\text{P}, \mathfrak{P})$ such that $D_{\mathcal{M}} = D'$. $\qquad \square$

We omit the proof since it is similar to the proof of Theorem 5.4.

Example 7.9 Consider a P2P data exchange system $\mathfrak{P}$:

P1: $\mathcal{R}(\text{P1}) = \{R^1(\cdot, \cdot)\}$, $D(\text{P1}) = \{R^1(a, null), R^1(s, t)\}$,

P2: $\mathcal{R}(\text{P2}) = \{R^2(\cdot, \cdot)\}$, $D(\text{P2}) = \{R^2(c, d), R^2(a, e)\}$,

$\Sigma(\text{P1}, \text{P2}) = \{\forall xy(R^2(x, y) \rightarrow \exists z R^1(x, z))\}$,

$trust = \{ (\text{P1}, less, \text{P2}) \}$.

The program $\Pi^{direct}(\text{P1}, \mathfrak{P})$ is

$dom(a)$. $\quad dom(b)$. $\quad \ldots \quad dom(u)$.

$R^1(a, null)$. $\quad R^1(s, t)$. $\quad R^2(c, d)$. $\quad R^2(a, e)$.

---

[4]The UDECs and RDECs in Σ are treated as UICs and RICs respectively and the condition of RIC-acyclic is checked using Definition 2.2.

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

$aux(x) \leftarrow R^1(x, null),\ \ not\ R^1_-(x, null, \mathbf{f_a}).$

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

$R^1_-(x, y, \mathbf{t^\star}) \leftarrow R^1_-(x, y, \mathbf{t_a}).$

$R^1_-(x, y, \mathbf{t^\star}) \leftarrow R^1(x, y).$

$R^1_-(x, y, \mathbf{f^\star}) \leftarrow R^1_-(x, y, \mathbf{f_a}).$

$R^1_-(x, y, \mathbf{f^\star}) \leftarrow dom(x), dom(y),\ \ not\ R^1(x, y).$

$R^1_-(x, y, \mathbf{t^{\star\star}}) \leftarrow R^1_-(x, y, \mathbf{t^\star}),\ \ not\ R^1_-(x, y, \mathbf{f_a}).$

$\leftarrow R^1_-(x, y, \mathbf{t_a}), R^1_-(x, y, \mathbf{f_a}).$

(Similarly for R^2)

The only neighborhood solution obtained from the program is $\{R^1(a, null), R^1(s, t),$ $R^2(c, d),\ R^2(a, e),\ R^1(c, null)\}$. The null value in $R^1(c, null)$ was used to restore consistency for the referential DEC. $\qquad\square$

7.3 The Transitive Case

It is natural to consider *transitive* DECs when a peer A that is being queried gets data from a peer B, which in its turn -and without A possibly knowing- gets data from a peer C to answer A's request. Most likely there is no explicit DEC from A to C. In order to approach peer consistent query answering for a peer P in this more complex scenario, it becomes necessary to integrate the data of all the peers that affect directly or indirectly peer P. This can be done in three different ways, leading to three possible semantics for the transitive case:

- **Semantics I:** Integrate the local specification of each of the peers that affect P. That is, consider that we have a unique database with the data in all the peers and that both DECs and ICs are considered traditional ICs, with preferences on how to repair to represent the trust relationships.

- **Semantics II:** Modify the local specification in such a way that the data in peer P, the *solutions* of the neighbors, and peer P's DECs and ICs are considered. Notice that this is a recursive definition. The base case corresponds to a peer with no DECs and therefore this solution semantics requires an acyclic accessibility graph.

- **Semantics III:** Modify the local specification in such a way that the data in peer P, the data from neighbors obtained as *peer consistent answers*, and peer P's DECs and ICs are considered. In this case, if P poses a query Q to a neighbor Q, what Q sends to P are not its plain answers to Q, but it PCAs to Q. This is also a recursive definition since in order to obtain the peer consistent answers of the neighbors we need the solutions under Semantics III of them. The base case corresponds to a peer with no DECs and therefore semantics III requires an acyclic accessibility graph.

We will define formally these three different semantics by means of stable models of logic programs in Section 7.3.1, 7.3.2 and 7.3.3, respectively. This is more natural and simpler than extending to the transitive case the definition of solution for the direct or local case. Of course, there might be no solutions. This fact is reflected in the absence of stable models for the logic program specification. A problematic case appears when there are implicit cyclic dependencies [Halevy *et al.*, 2003].

In order to define the *global solutions* for a peer in the transitive case, we need to determine which are the peers in $\mathcal{P}$ that may affect the data in it. Now a peer is not only affected by its neighbors, but also by the neighbors of its neighbors and so on. Therefore the relevant peers to determine the solution for a peer P are those in $\mathcal{AC}(\mathtt{P})$.

The presence of cycles, either through constraints or trust relationships, have an impact on the semantics. There is a cycle through constraints if the set $\Sigma \cup IC$ is RIC-cyclic. If that is the case, semantics I, II and III are not defined since they might give some solutions which are counterintuitive (see Example 5.12).

There is a cycle through trust relationships if the accessibility graph $\mathcal{G}_A(\mathfrak{P})$ is cyclic. Since semantics II and III are inductively defined, a cycle in $\mathcal{G}_A(\mathfrak{P})$ would result in an infinite loop. In Sections 7.3.2 and 7.3.3 we will show how to avoid the infinite loops and return an error in this case.

In the rest of the chapter we assume that we can enforce the satisfaction of RDECs by adding tuples with *null*, that $null \in \mathcal{U}$ and that the peer instances might contain *null*. The next three sections formalize semantics I, II and III for solutions in the transitive case.

7.3.1 Solutions under Semantics I

The first semantics for solutions that we will define consists in considering that we have a unique database with the data in all the peers and that the DECs and ICs are both considered as traditional ICs, with preferences on how to repair (to represent the trust relationships). Therefore, the global solution for a peer P will be an instance for the peers accessible from P that respects the ICs and the DECs and trust relationships between them and stays "as close as possible" to $\bar{D}(\mathcal{AC}(\text{P}))$. Respecting the trust relationships implies that, if P1 trusts itself less than P2, any virtual modification to enforce the DECs has to be done to peer P1. On the other hand, if P1 trusts peer P2 as much as itself, the modifications can be done at any of the peers. Finally, if P1 trusts itself more than P2, then that DEC or peer is not relevant for finding the solution for P1.

In this case, finding the global solution for a peer P given a set of DECs and

trust relationships can be seen as repairing the set of data in all the peers accessible from P with respect to the DECs (seen as ICs) and the local ICs, where the repairs are obtained taking into consideration the trust relationships. Based on the repair program defined in Chapter 5, we will can define a *global solution program* to compute the solutions of a peer for RDECs and UDECs. In the definition we use $\mathcal{A}(\psi)$ to denote the relevant attributes of a DEC ψ, which are defined as for ICs (see Definition 4.2).

Definition 7.11 Given a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$ with only UDECs and RDECs in Σ , the *solution program for semantic* I for a peer $P \in \mathcal{P}$, denoted $\Pi^I(P, \mathfrak{P})$, contains the following facts and rules:

1. $dom(x)$, for every $x \in (\mathcal{U} \smallsetminus \{null\})$.

2. $R(\bar{a})$, for each atom $R(\bar{a}) \in \bar{D}(\mathcal{AC}(P))$.

3. For every UDEC $\psi \in \Sigma(\text{Pi}, \text{Pj})$ of the form (7.5) such that $\text{Pi}, \text{Pj} \in \mathcal{AC}(P)$, and there exists $(\text{Pi}, \{same \text{ or } less\}, \text{Pj}) \in trust$, the rules:

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

Here, $\mathcal{A}(\psi)$ is the set of relevant attributes for ψ and $\bar{\varphi}$ is a conjunction of built-ins that is equivalent to the negation of φ. R_{Pi} is defined as follows for $\mathcal{R} = \{R_i \mid i = 1, \ldots, n\}$:

$$R_{\text{Pi}} = \begin{cases} \mathcal{R} \cap \mathcal{R}(\text{Pi}) & \text{if } (\text{Pi}, less, \text{Pj}) \in trust \\ \mathcal{R} & \text{if } (\text{Pi}, same, \text{Pj}) \in trust \end{cases}$$

Q_P is defined analogously.

4. For every RDEC $\psi \in \Sigma(\text{Pi}, \text{Pj})$ of the form (7.6) such that $\text{Pi}, \text{Pj} \in \mathcal{AC}(P)$:

   (a) If $(\text{Pi}, same, \text{Pj}) \in trust$, the rules:

   $$R_{\_}(\bar{x}, \mathbf{f_a}) \vee Q_{\_}(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow P_{\_}(\bar{x}, \mathbf{t^\star}), \ not \ aux_\psi(\bar{x}'), \bar{x}' \neq null.$$

(b) If $(\mathtt{Pi}, \mathit{less}, \mathtt{Pj}) \in \mathit{trust}$ and $R \in \mathcal{R}(\mathtt{Pi})$, the rules:

$$R_{\_}(\bar{x}, \mathbf{f_a}) \leftarrow P_{\_}(\bar{x}, \mathbf{t}^{\star}),\ \mathit{not}\ \mathit{aux}_{\psi}(\bar{x}'), \bar{x}' \neq \mathit{null}.$$

(c) If $(\mathtt{Pi}, \mathit{less}, \mathtt{Pj}) \in \mathit{trust}$ and $Q \in \mathcal{R}(\mathtt{Pi})$, the rules:

$$Q_{\_}(\bar{x}', \overline{\mathit{null}}, \mathbf{t_a}) \leftarrow P_{\_}(\bar{x}, \mathbf{t}^{\star}),\ \mathit{not}\ \mathit{aux}_{\psi}(\bar{x}'), \bar{x}' \neq \mathit{null}.$$

Plus the auxiliary rules:

$$\mathit{aux}_{\psi}(\bar{x}') \leftarrow Q(\bar{x}', \overline{\mathit{null}}),\ \ \mathit{not}\ Q_{\_}(\bar{x}', \overline{\mathit{null}}, \mathbf{f_a}),\ \bar{x}' \neq \mathit{null}.$$

$$\mathit{aux}_{\psi}(\bar{x}') \leftarrow Q_{\_}(\bar{x}', \bar{y}, \mathbf{t}^{\star}),\ \ \mathit{not}\ Q_{\_}(\bar{x}', \bar{y}, \mathbf{f_a}),\ \bar{x}' \neq \mathit{null}, y_i \neq \mathit{null}. \qquad \text{for every}\ \ y_i \in \bar{y}$$

5. For every UIC $\psi \in \mathit{IC}(\mathtt{Pi})$ of the form (2.2) and such that $\mathtt{Pi} \in \mathcal{AC}(\mathtt{P})$, the rules:

$$\bigvee_{i=1}^{n} P_{i\_}(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} Q_{j\_}(\bar{y}_j, \mathbf{t_a}) \ \leftarrow\ \bigwedge_{i=1}^{n} P_{i\_}(\bar{x}_i, \mathbf{t}^{\star}), \bigwedge_{j=1}^{m} Q_{j\_}(\bar{y}_j, \mathbf{f}^{\star}), \bigwedge_{x_l \in \mathcal{A}(\psi)} x_l \neq \mathit{null}, \bar{\varphi}.$$

Here, $\mathcal{A}(\psi)$ is the set of relevant attributes for ψ, and $\bar{\varphi}$ is a conjunction of built-ins that is equivalent to the negation of φ.

6. For every RIC $\psi \in \mathit{IC}(\mathtt{Pi})$ of the form (2.3) and such that $\mathtt{Pi} \in \mathcal{AC}(\mathtt{P})$, the rules:

$$P_{\_}(\bar{x}, \mathbf{f_a}) \vee Q_{\_}(\bar{x}', \overline{\mathit{null}}, \mathbf{t_a}) \leftarrow P_{\_}(\bar{x}, \mathbf{t}^{\star}),\ \mathit{not}\ \mathit{aux}_{\psi}(\bar{x}'), \bar{x}' \neq \mathit{null}.$$
$$\mathit{aux}_{\psi}(\bar{x}') \leftarrow Q(\bar{x}', \overline{\mathit{null}}),\ \ \mathit{not}\ Q_{\_}(\bar{x}', \overline{\mathit{null}}, \mathbf{f_a}), \bar{x}' \neq \mathit{null}.$$

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

$$\mathit{aux}_{\psi}(\bar{x}') \leftarrow Q_{\_}(\bar{x}', \bar{y}, \mathbf{t}^{\star}),\ \ \mathit{not}\ Q_{\_}(\bar{x}', \bar{y}, \mathbf{f_a}), \bar{x}' \neq \mathit{null}, y_i \neq \mathit{null}.$$

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

$$R_{\_}(\bar{x}, \mathbf{f}^{\star}) \leftarrow \mathit{dom}(\bar{x}),\ \mathit{not}\ R(\bar{x}).$$
$$R_{\_}(\bar{x}, \mathbf{f}^{\star}) \leftarrow R_{\_}(\bar{x}, \mathbf{f_a}).$$
$$R_{\_}(\bar{x}, \mathbf{t}^{\star}) \leftarrow R(\bar{x}).$$
$$R_{\_}(\bar{x}, \mathbf{t}^{\star}) \leftarrow R_{\_}(\bar{x}, \mathbf{t_a}).$$

8. For each predicate $R \in \mathcal{R}(\mathcal{AC}(\mathtt{P}))$, the interpretation rule:

$R\_(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow R\_(\bar{x}, \mathbf{t}^{\star}), \;\; not \; R\_(\bar{x}, \mathbf{f_a}).$

9. For each predicate $R \in \mathcal{R}(\mathcal{AC}(\mathtt{P}))$, the program denial constraint:

$\leftarrow \; R\_(\bar{x}, \mathbf{t_a}), \; R\_(\bar{x}, \mathbf{f_a}). \hspace{3cm} \square$

This logical program specifies the *global solutions* for peer $\mathtt{P}$ taking into consideration the transitive relations it has with other peers. The DECs and local ICs of all peers that are accessible from peer $\mathtt{P}$ have to be satisfied by the *global solution* that is captured by the atoms with annotation constant $\mathbf{t}^{\star\star}$. This program is very similar to the repair programs from Chapter 5. In fact, rules in 5., 6., 7., 8. and 9 are the same as the ones to deal with ICs in the repair program from Definition 5.2. Rules in 3 and 4 enforce the satisfaction of the DECs by only modifying the peers that are less or equally trusted. We also use the same annotation constants. The use of atoms with annotation constant $\mathbf{f}^{\star}$ and *dom* atoms can be avoided by using the same optimization presented in Appendix A. Here, we used them because it makes the program easier to understand.

Definition 7.12 The *P2P instance* associated to a stable model $\mathcal{M}$ of program $\Pi^{\mathrm{I}}(\mathtt{P}, \mathfrak{P})$ is $D_{\mathcal{M}} = \{R(\bar{a}) \mid R(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M} \text{ and } R \in \mathcal{R}(\mathcal{AC}(\mathtt{P}))\}. \hspace{1cm} \square$

Definition 7.13 (transitive case, semantics $\mathtt{I}$) Given a peer $\mathtt{P}$ in a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$ such that $\Sigma \cap IC$ is RIC-acyclic, an instance D over $\mathcal{R}(\mathcal{A}(\mathtt{P}))$ is a *global solution under semantics* $\mathtt{I}$ for $\mathtt{P}$ if there exists a stable model $\mathcal{M}$ of $\Pi^{\mathrm{I}}(\mathtt{P}, \mathfrak{P})$ such that $D = D_{\mathcal{M}}$. $S_G(\mathtt{P})$ is the set of global solutions for peer $\mathtt{P}$.

An instance s is a *solution under semantics* $\mathtt{I}$ for peer $\mathtt{P}$ if there is a global solution D for $\mathtt{P}$ such that $s = D|\mathtt{P}$. $S(\mathtt{P})$ is the set of solutions for $\mathtt{P}$. As in the local case, $\mathtt{P}$'s *peer consistent answers* (PCA) are those answers that can be retrieved from *every* solution for $\mathtt{P}$. $\hspace{1cm} \square$

Example 7.10 (example 7.3 continued) $\Pi^{I}(\text{P1}, \mathfrak{P})$:

$dom(a)$.　　$dom(b)$.　　$\ldots$　　$dom(u)$.

$R^1(a,b)$.　　$R^1(s,t)$.　　$R^2(c,d)$.　　$R^2(a,e)$.　　$R^3(s,u)$.

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

$R^1_{\_}(x,y,\mathbf{f_a}) \leftarrow R^1_{\_}(x,y,\mathbf{t^\star}), R^3_{\_}(x,z,\mathbf{t^\star}), y \neq z, x \neq null, y \neq null, z \neq null$.

$\left.\begin{array}{l} R^1_{\_}(x,y,\mathbf{t^\star}) \leftarrow R^1_{\_}(x,y,\mathbf{t_a}). \\[4pt] R^1_{\_}(x,y,\mathbf{t^\star}) \leftarrow R^1(x,y). \\[4pt] R^1_{\_}(x,y,\mathbf{f^\star}) \leftarrow R^1_{\_}(x,y,\mathbf{f_a}). \\[4pt] R^1_{\_}(x,y,\mathbf{f^\star}) \leftarrow dom(x), dom(y),\ not\ R^1(x,y). \\[4pt] R^1_{\_}(x,y,\mathbf{t^{\star\star}}) \leftarrow R^1_{\_}(x,y,\mathbf{t^\star}),\ not\ R^1_{\_}(x,y,\mathbf{f_a}). \\[4pt] \leftarrow R^1_{\_}(x,y,\mathbf{t_a}), R^1_{\_}(x,y,\mathbf{f_a}). \end{array}\right\}$ (Similarly for R^2 and R^3)

The inconsistencies with respect to the DECs are restored by virtually modifying only peer P1. By running the program in DLV, we get the only stable model:

$\mathcal{M} = \{\ldots, R^1(a,b),\ R^1(a,b,\mathbf{t^\star}),\ R^1(s,t),\ R^1(s,t,\mathbf{t^\star}),\ R^1(s,t,\mathbf{f_a}),\ R^2(a,e),\ R^2(c,d),$
$R^2(a,e,\mathbf{t^\star}),\ R^2(c,d,\mathbf{t^\star}),\ R^3(s,u),\ R^3(s,u,\mathbf{t^\star}),\ \underline{R^3(s,u,\mathbf{t^{\star\star}})},\ \underline{R^2(a,e,\mathbf{t^{\star\star}})},\ \underline{R^2(c,d,\mathbf{t^{\star\star}})},$
$R^1(a,e,\mathbf{t_a}),\ R^1(c,d,\mathbf{t_a}),\ R^1(a,e,\mathbf{t^\star}),\ R^1(c,d,\mathbf{t^\star}),\ \underline{R^1(a,e,\mathbf{t^{\star\star}})},\ \underline{R^1(c,d,\mathbf{t^{\star\star}})},\ \underline{R^1(a,b,}$
$\underline{\mathbf{t^{\star\star}})}\}$.

$S_G(\text{P}) = \{\{R^3(s,u),\ R^2(a,e),\ R^2(c,d), R^1(a,e),\ R^1(c,d),\ R^1(a,b)\}\}$,

$S(\text{P}) = \{\{R^1(a,e),\ R^1(c,d),\ R^1(a,b)\}\}$.

In this case, since $\mathcal{N}(\text{P1}) = \mathcal{AC}(\text{P1})$, the local solution coincides with the transitive solution. Finally, for the query $Q_{\text{P}}(x) \leftarrow R^1(x,y)$ posed to P, the peer consistent answers are $PCA(Q_{\text{P}}) = \{(a), (c)\}$.　　　　　$\square$

Example 7.11 (example 7.2 continued) Consider the following instances of peers P1, P2 and P3 : $D(\text{P1}) = \{R^1(a,2)\}$, $D(\text{P2}) = \{R^2(c,4), R^2(d,5)\}$, and $D(\text{P3}) = \{R^3(c,4)\}$. The solution program $\Pi^{I}(\text{P1}, \mathfrak{P})$ is:

$dom(a). \quad dom(c). \quad \ldots$

$R^1(a,2). \quad R^2(c,4). \quad R^2(d,5). \quad R^3(c,4).$

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

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

$$\left.\begin{array}{l} R^1_\_(x,y,\mathbf{t^\star}) \leftarrow R^1_\_(x,y,\mathbf{t_a}). \\[4pt] R^1_\_(x,y,\mathbf{t^\star}) \leftarrow R^1(x,y). \\[4pt] R^1_\_(x,y,\mathbf{f^\star}) \leftarrow R^1_\_(x,y,\mathbf{f_a}). \\[4pt] R^1_\_(x,y,\mathbf{f^\star}) \leftarrow dom(x), dom(y), \; not \; R^1(x,y). \\[4pt] R^1_\_(x,y,\mathbf{t^{\star\star}}) \leftarrow R^1_\_(x,y,\mathbf{t^\star}), \; not \; R^1_\_(x,y,\mathbf{f_a}). \\[4pt] \leftarrow R^1_\_(x,y,\mathbf{t_a}), R^1_\_(x,y,\mathbf{f_a}). \end{array}\right\} \text{(Similarly for } R^2 \text{ and } R^3)$$

The inconsistencies with respect to the DECs in $\Sigma(\mathtt{P1},\mathtt{P2})$ are restored by virtually modifying only peer $\mathtt{P1}$. Those with respect to DECs in $\Sigma(\mathtt{P2},\mathtt{P3})$ are solved by modifying both $\mathtt{P2}$ and $\mathtt{P3}$. The global solutions are $S^{\mathtt{I}}_G(\mathtt{P1}) = \{\{R^1(a,2), R^2(c,4), R^2(d,5), R^1(c,4), R^1(d,5)\}, \{R^1(a,2), R^2(d,5), R^3(c,4), R^1(d,5)\}\}$. Then, $S^{\mathtt{I}}(\mathtt{P1}) = \{\{R^1(a,2), R^1(c,4), R^1(d,5)\}, \{R^1(a,2), R^1(d,5)\}\}$. $\qquad\square$

Example 7.12 Consider the P2P data exchange system $\mathfrak{P}$ with trust relationships as shown in Figure 7.3.

> $\mathtt{P1}$: $\mathcal{R}(\mathtt{P1}) = \{R^1\}, \;\; D(\mathtt{P1}) = \{R^1(c,d), R^1(f,g)\},$
>
> $\mathtt{P2}$: $\mathcal{R}(\mathtt{P2}) = \{R^2\}, \;\; D(\mathtt{P2}) = \{R^2(c,d), R^2(a,e)\},$
>
> $\Sigma(\mathtt{P1},\mathtt{P2})$: $\forall x \forall y (R^2(x,y) \rightarrow R^1(x,y)),$
>
> $trust=\{(\mathtt{P1},same,\mathtt{P2})\}.$
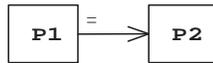
Figure 7.3: Accessibility graph of Example 7.12

$\mathtt{P1}$ trusts its own data as much as it trusts $\mathtt{P2}$'s data. If we want the solution for

P1, we need to check if its DECs are being satisfied. If they are not, the conflicts can be solved by modifying the data in P1 or in P2. Here, $\Sigma(\texttt{P1}, \texttt{P2})$ is not satisfied, because $(a, e) \in R^2$, but it does not belong to R^1. In order to restore consistency with a minimal set of modifications, we can add (a, e) from R^2 or add it to R^1. This alternatives are dealt with in $\Pi^{\mathtt{I}}(\texttt{P1}, \mathfrak{P})$ through the rule:

$$R_-^2(x, y, \mathbf{f_a}) \vee R_-^1(x, y, \mathbf{t_a}) \leftarrow R_-^2(x, y, \mathbf{t}^\star), R_-^1(x, y, \mathbf{f}^\star), x \neq null, y \neq null.$$

From the stable models of the program we get $S_{\mathcal{G}}(\texttt{P1}) = \{\{R^1(c, d), R^1(f, g), R^1(a, e), R^2(c, d), R^2(a, e)\}, \{R^1(c, d), R^1(f, g), R^2(c, d)\}\}$. By projecting the elements in $S_{\mathcal{G}}(\texttt{P1})$ onto P1's schema, we get $S(\texttt{P1}) = \{\{R^1(c, d), R^1(f, g), R^1(a, e)\}, \{R^1(c, d), R^1(f, g)\}\}$.

The peer consistent answers to a query posed to P1 are the answers we will get from all the different solutions. For query $Q : R^1(c, y)$, we get (d) from both the first and second solution for P1, therefore (d) is the peer consistent answer for Q.

For P2, $\mathcal{AC}(\texttt{P2}) = \{P2\}$, and since P2 has no local ICs, the solution for P2 corresponds exactly to the data already stored in it. □

Example 7.13 (example 7.9 continued) For this P2P system, the program $\Pi^{\mathtt{I}}(\texttt{P1}, \mathfrak{P})$ coincides with $\Pi^{direct}(\texttt{P1}, \mathfrak{P})$ since $\mathcal{N}(\texttt{P1}) = \mathcal{AC}(\texttt{P1})$. □

Computation of PCA under Semantics I

In order to give peer consistent answers under semantics I, each peer in the system should be capable of answering queries from users (user-queries) and from peers (peer-queries) and process each of them in a different way [5]. Answers to user-queries are peer consistent answers and answers to peer-queries are data, mappings, trust relationships and ICs.

---

[5] The differentiation between user and peer queries is also used in [Calvanese *et al.*, 2004b; Calvanese *et al.*, 2005].

The following procedure is a naive implementation of semantics I: when a user-query is posed to a peer P, it will need to get all the data, mappings and ICs of all the peers accessible from it. As we will later show, this data can be obtained by the peer by sending a peer-query to its neighbors which will query their neighbors and so for. They will return their data, mappings, trust relationships and ICs. This information is then used in peer P to compute all its local solutions. The solutions are then used to get the peer consistent answers to the user-query and are returned to the user.

Example 7.14 (example 7.11 continued) Consider a user-query $Q : R^1(x, y)$ posed to peer P1. In order to answer the query we need the information of all the accessible peers. Peer P1 send a peer-query to its only neighbor P2 requesting all its information and the ones of its neighbors. Then, peer P2 send a peer-query to P3 which has no neighbors and therefore returns only its own data (if it had had ICs it would have also returned them). Peer P2 gets the answer from P3 and sends to peer P1 the data of P3 and the mappings, trust relationships and data of P2. Now, P1 has all the data needed to write $\Pi(\text{P1}, \mathfrak{P})$ as shown in Example 7.11. The peer consistent answers to the user-query Q are $\{(a, 2), (d, 5)\}$. □

In this process we need to have a way to deal with cycles of the accessibility graph. A technique used in a similar context consists of using a unique code [Calvanese *et al.*, 2004a]. All the peer-queries sent as a consequence of the same user-query should be marked with a unique code so that if a peer receives a query with a code that it has already processed, i.e., there is a cycle, it gives an empty answer to the query it received. The following example shows how to answer a query in the presence of a cycle in the accessibility graph.

Example 7.15 Consider a P2P data exchange system with a cycle in the accessibility graph (see Figure 7.4):

P1: $\mathcal{R}(\mathtt{P1}) = \{R^1(\cdot, \cdot), S^1(\cdot)\},\ \ D(\mathtt{P1}) = \{R^1(2, b), S^1(a), S^1(b), S^1(h)\},$

P2: $\mathcal{R}(\mathtt{P2}) = \{R^2(\cdot, \cdot)\},\ \ D(\mathtt{P2}) = \{R^2(a, 6), R^2(b, 4), R^2(c, 6)\},$

P3: $\mathcal{R}(\mathtt{P3}) = \{R^3(\cdot, \cdot)\},\ \ D(\mathtt{P3}) = \{R^3(a, 6), R^3(h, 8), R^3(g, 2), R^3(b, 4), R^3(a, 1)\},$

$\Sigma(\mathtt{P1}, \mathtt{P2}) = \{\forall xy(R^2(x, y) \rightarrow \exists z R^1(z, x))\},$

$\Sigma(\mathtt{P2}, \mathtt{P3}) = \{\forall xy(R^2(x, y) \rightarrow R^3(x, y))\},$

$\Sigma(\mathtt{P3}, \mathtt{P1}) = \{\forall xy(R^3(x, y) \rightarrow S^1(x))\},$

$trust = \{\ (\mathtt{P1}, less, \mathtt{P2}),\ (\mathtt{P2}, less, \mathtt{P3}),\ (\mathtt{P3}, less, \mathtt{P1})\ \}.$
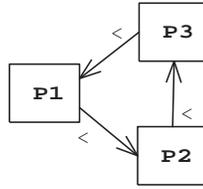
Figure 7.4: Accessibility graph of Example 7.15

If the user-query $Q_0 : \exists x R^1(x, y)$ is posed to peer P1, it will need to create a unique *ID* for the query, e.g. *A11*, and send a peer query to P2 requesting its DECs, trust relationships, ICs and data and the ones of its accessible peers (see Figure 7.5(a)). Peer P2 would first check if a query with $ID = A11$ has already been processed. Since it has not, it will proceed to send a peer-query to P3 requesting their information (see Figure 7.5(b)). After receiving the peer-query, P3 will check if it has processed a query with $ID = A11$. Since it has not, it will send a request to its only neighbor, peer P1, a request for its mapping, constraints and data (see Figure 7.5(c)). Now, P1 will check if it has processed a query with $ID = A11$. Since it has, it will not send any peer-queries to its neighbors, and it will send an empty answer to P3 (see Figure 7.5(d)). Now P3 will send to peer P2: $\Sigma(\mathtt{P3}, \mathtt{P1})$, $(\mathtt{P3}, less, \mathtt{P1})$ and $D(\mathtt{P3})$ (see Figure 7.5(e)). Now P2 has all the needed information to reply to peer P1, and will send $\Sigma(\mathtt{P3}, \mathtt{P1})$, $IC(\mathtt{P3})$, $(\mathtt{P3}, less, \mathtt{P1})$, $D(\mathtt{P3})$, $\Sigma(\mathtt{P2}, \mathtt{P3})$, $(\mathtt{P2}, less, \mathtt{P3})$ } and $D(\mathtt{P2})$ (see Figure 7.5(f)).

(a) P1 leaves the user-query pending and send a peer-query to P2

(b) P2 leaves the peer-query from P1 pending and send a peer-query to P3

(c) P3 leaves the peer-query from P2 pending and send a peer-query to P1

(d) P1 sends an empty answer to P3

(e) P3 sends an answer containing its DECs, ICs, trust relationships and data to P2

(f) P2 sends an answer containing the DECs, ICs, trust relationships and data of itself and of P3 to P1

Figure 7.5: Dealing with cycles in the accessibility graph

Peer P1 has now all the needed information to write $\Pi^I(\texttt{P1}, \mathfrak{P})$. The rules that enforce the satisfaction of the DECs are:

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

$$aux(x) \leftarrow R^1_-(y, x, \mathbf{t^\star}), x \neq null, y \neq null.$$

$$aux(x) \leftarrow R^1(null, x), \;\; not \; R^1_-(null, x, \mathbf{f_a}), x \neq null.$$

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

$$R^3_-(x, y, \mathbf{f_a}) \leftarrow R^3_-(x, y, \mathbf{t^\star}), S^1_-(x, \mathbf{f^\star}), x \neq null.$$

The program $\Pi^I(\texttt{P1}, \mathfrak{P})$ has only one stable model: $\mathcal{M} = \{R^2_-(c, 6, \mathbf{f_a}), aux(b), R^3_-(g, 2, \mathbf{f_a}), S^1_-(b, \mathbf{t^{\star\star}}), S^1_-(a, \mathbf{t^{\star\star}}), S^1_-(h, \mathbf{t^{\star\star}}), R^3_-(b, 4, \mathbf{t^{\star\star}}), R^3_-(a, 1, \mathbf{t^{\star\star}}), R^3_-(a, 6, \mathbf{t^{\star\star}}), R^3_-(h, 8, \mathbf{t^{\star\star}}), R^2_-(b, 4, \mathbf{t^{\star\star}}), R^2_-(a, 6, \mathbf{t^{\star\star}}), R^1_-(null, a, \mathbf{t_a}), R^1_-(null, c, \mathbf{t_a}), R^1_-(2, b, \mathbf{t^{\star\star}}), R^1_-(null, a, \mathbf{t^{\star\star}}), R^1_-(null, c, \mathbf{t^{\star\star}})\}$. Thus, there is one solution for peer P1, $S(\texttt{P1}) = \{\{S^1(b), S^1(a), S^1(h), R^1(2, b), R^1(null, a), R^1(null, c)\}\}$. The peer consistent answers for Q_0

are $\{(a), (b), (c)\}$. □

Example 7.16 Consider the following P2P data exchange system:

P1: $\mathcal{R}(\text{P1}) = \{R^1\}, \ D(\text{P1}) = \{R^1(a, b)\}$,

P2, $\mathcal{R}(\text{P2}) = \{R^2\}: \ D(\text{P2}) = \{\}$,

$\Sigma(\text{P1}, \text{P2}) = \Sigma(\text{P2}, \text{P1}): \ \{\forall xy(R^1(x, y) \ \rightarrow \ R^2(x, y))\}$.
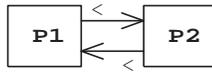
Figure 7.6: Accessibility graph of Example 7.16

If peer P1 receives a user-query, it will need to send a query to P2 to get its DECs, ICs, trust relationships and data. Peer P2 will send a query to P1 to get its data. Since the query that P1 received from P2 has the same unique ID, P1 will provide an empty answer to P2. Peer P2 will now send $\Sigma(\text{P2}, \text{P1})$, $(\text{P2}, \textit{less}, \text{P1})$ and $D\text{P2}$ to peer P1. Now P1 has all the information to write $\Pi^{\text{I}}(\text{P1}, \mathfrak{P})$.

If a user-query is posed to P2, the process would be analogous. The rules to enforce the satisfaction of DECs in programs $\Pi^{\text{I}}(\text{P1}, \mathfrak{P})$ and $\Pi^{\text{I}}(\text{P2}, \mathfrak{P})$ are the same and equal to:

$$R^2(x, y, \mathbf{t_a}) \ \leftarrow \ R^1(x, y, \mathbf{t}^\star), R^2(x, y, \mathbf{f}^\star), x \neq \textit{null}, y \neq \textit{null}.$$
$$R^2(x, y, \mathbf{f_a}) \ \leftarrow \ R^1(x, y, \mathbf{t}^\star), R^2(x, y, \mathbf{f}^\star), x \neq \textit{null}, y \neq \textit{null}.$$

Using $\Pi^{\text{I}}(\text{P1}, \mathfrak{P})$ and $\Pi^{\text{I}}(\text{P2}, \mathfrak{P})$ we can compute $S(\text{P1}) = \{\}$, and $S(\text{P2}) = \{R^2(a, b)\}$. □

The implementation of the semantics can be optimized in several ways. For example we can reduce the amount of data sent between peers by sending only the data of the

accessible peers that is relevant to answer the user-query and check the satisfaction of the relevant DECs.

Example 7.17 (example 7.15 continued). If a user poses query $Q_1 : S^1(x)$ to peer P1, there is no need to use program $\Pi^{\mathrm{I}}(\text{P1}, \mathfrak{P})$ to obtain the peer consistent answers. This is because none of the DECs in P1 are related to S^1 and therefore the data in the rest of the peers will have no impact on the data in S^1. The peer consistent answers can be obtained by directly querying the $D(\text{P1})$. □

Other optimization consist can be based on caching part of the data, solutions or partial solutions. Several optimization used in the context of CQA can be used for PDM [Caniupan, 2006].

7.3.2 Solutions under Semantics II

The second semantics consists in modifying the local specification in such a way that it considers the data in peer P, the *solutions* (under the same semantics) of the neighboring peers and peer P's DECs and ICs. In this semantics, the transitive information is gathered in the solutions of the neighboring peers.

Example 7.18 (example 7.2 and 7.11 continued) Figure 7.7 shows the accessibility graphs for peers P1, P2 and P3. Under solution semantics II, the solutions for peer P1 will only be affected by its own data, the solutions of peer P2, the DECs between them and the ICs of P1. Therefore, in order to compute the solutions for peer P1 the peer will need to request the solutions of peer P2. Now, P2 will need the solutions of P3. Peer P2 will request the solutions of P3. Since P3 has no DECs nor ICs, its solution is $D(\text{P3})$. This solution will be sent back to P2. Having the solution of P3, peer P2 is able to compute its own solutions and send them to P1. It might be the

case that peer P2 returns more than one solution to P1. If that is the case, $S(\text{P1})$ will be the union of the solutions obtained by using, one-by-one the solutions of P2. □

(a) $\mathcal{G}_A(\mathfrak{P})$ restricted to $\mathcal{AC}(\text{P1})$ (b) $\mathcal{G}_A(\mathfrak{P})$ restricted to $\mathcal{AC}(\text{P2})$ (c) $\mathcal{G}_A(\mathfrak{P})$ restricted to $\mathcal{AC}(\text{P3})$
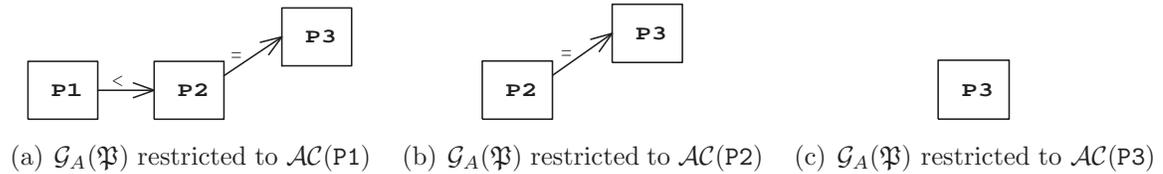
Figure 7.7: Accessibility graphs of Example 7.18

Note, that as described in Example 7.18, a peer P might receive more than one solution from each neighboring peer. In that case, the solutions of P have to be computed by taking all the possible combinations of solutions of the neighbors (using one solutions from each peer).

If we have a peer data exchange system with cycles through trust relationships, it would not be possible to directly apply this semantics for solutions, as shown in the next example.

Example 7.19 Consider a P2P data exchange system with the following DECs:

$\Sigma(\text{P1}, \text{P2})\colon \ \forall xy(R^2(x,y) \ \rightarrow \ R^1(x,y)),$

$\Sigma(\text{P2}, \text{P3})\colon \ \forall xy(R^3(x,y) \ \rightarrow \ R^2(x,y)),$

$\Sigma(\text{P3}, \text{P1})\colon \ \forall xy(R^1(x,y) \ \rightarrow \ \exists z R^3(x,z)).$
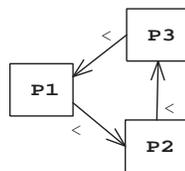
Figure 7.8: Accessibility graph of Example 7.19

The trust relationships for these peers are shown in Figure 7.8. If we want the solutions of peer P1, the situation would be the following: $S(\text{P1})$ needs $S(\text{P2})$, $S(\text{P2})$ needs $S(\text{P3})$, $S(\text{P3})$ needs $S(\text{P1})$, $S(\text{P1})$ needs $S(\text{P2})$, ... □

Definition 7.14 Consider a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$, a peer $P \in \mathcal{P}$, and a set $\mathcal{S} = \{s_{P1}, \ldots, s_{Pn}\}$, where $\mathcal{N}(P) = \{P, P1, \ldots, Pn\}$ and s_{Pj} is a database with the same schema as Pj. The *solution program for semantic* II, $\Pi^{II}(P, \mathcal{S}, \mathfrak{P})$ is:

1. $dom(x)$, for every $x \in (\mathcal{U} \setminus \{null\})$.

2. $R(\bar{a})$, for each atom $R(\bar{a}) \in \bar{D}(P)$.

3. $R(\bar{a})$, for each $R(\bar{a}) \in s$ with $s \in \mathcal{S}$.

4. For every UDEC $\psi \in \Sigma(P, Pj)$ of the form (7.5) such that $Pj \in \mathcal{N}(P)$ and there exists an $(P, \{same \text{ or } less\}, Pj) \in trust$, the rules:

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

Here, $\mathcal{A}(\psi)$ is the set of relevant attributes for ψ, and $\bar{\varphi}$ is a conjunction of built-ins that is equivalent to the negation of φ. R_P is defined as follows, for $\mathcal{R} = \{R_i \mid i \in \{1, \ldots, n\}\}$:

$$R_P = \begin{cases} \mathcal{R} \cap \mathcal{R}(P), & \text{if } (P, less, Pj) \in trust \\ \mathcal{R}, & \text{if } (P, same, Pj) \in trust \end{cases}$$

Q_P is defined analogously.

5. For every RDEC $\psi \in \Sigma(P, Pj)$ of the form (7.6) such that $Pj \in \mathcal{N}(P)$ and there exists an $(P, \{same \text{ or } less\}, Pj) \in trust$:

   (a) If $(P, same, Pj) \in trust$, the rules:

   $$R\text{-}(\bar{x}, \mathbf{f_a}) \vee Q\text{-}(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow P\text{-}(\bar{x}, \mathbf{t^\star}), \ not \ aux_\psi(\bar{x}'), \bar{x}' \neq null.$$

   (b) If $(P, less, Pj) \in trust$ and $R \in \mathcal{R}(P)$, the rules:

   $$R\text{-}(\bar{x}, \mathbf{f_a}) \leftarrow P\text{-}(\bar{x}, \mathbf{t^\star}), \ not \ aux_\psi(\bar{x}'), \bar{x}' \neq null.$$

(c) If $(\mathtt{P}, \textit{less}, \mathtt{Pj}) \in \textit{trust}$ and $Q \in \mathcal{R}(\mathtt{P})$, the rules:

$$Q_-(\bar{x}', \overline{\textit{null}}, \mathbf{t_a}) \leftarrow P_-(\bar{x}, \mathbf{t}^\star), \ \textit{not } aux_\psi(\bar{x}'), \bar{x}' \neq \textit{null}.$$

Plus the auxiliary rules:

$$aux_\psi(\bar{x}') \leftarrow Q(\bar{x}', \overline{\textit{null}}), \ \textit{not } Q_-(\bar{x}', \overline{\textit{null}}, \mathbf{f_a}), \bar{x}' \neq \textit{null},$$

$$aux_\psi(\bar{x}') \leftarrow Q_-(\bar{x}', \bar{y}, \mathbf{t}^\star), \ \textit{not } Q_-(\bar{x}', \bar{y}, \mathbf{f_a}), \bar{x}' \neq \textit{null}, y_i \neq \textit{null}, \qquad \text{for every } y_i \in \bar{y}.$$

6. For every UIC $\psi \in IC(\mathtt{Pi})$ of the form (2.2) and such that $\mathtt{Pi} \in \mathcal{N}(\mathtt{P})$, the rules:

$$\bigvee_{i=1}^{n} P_{i\text{-}}(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} Q_{j\text{-}}(\bar{y}_j, \mathbf{t_a}) \leftarrow \bigwedge_{i=1}^{n} P_{i\text{-}}(\bar{x}_i, \mathbf{t}^\star), \bigwedge_{j=1}^{m} Q_{j\text{-}}(\bar{y}_j, \mathbf{f}^\star), \bigwedge_{x_l \in \mathcal{A}(\psi)} x_l \neq \textit{null}, \bar{\varphi}.$$

Here, $\mathcal{A}(\psi)$ is the set of relevant attributes for ψ, and $\bar{\varphi}$ is a conjunction of built-ins that is equivalent to the negation of φ.

7. For every $\psi \in IC(\mathtt{Pi})$ of the form (2.3) and such that $\mathtt{Pi} \in \mathcal{N}(\mathtt{P})$, the rules:

$$P_-(\bar{x}, \mathbf{f_a}) \vee Q_-(\bar{x}', \overline{\textit{null}}, \mathbf{t_a}) \leftarrow P_-(\bar{x}, \mathbf{t}^\star), \ \textit{not} \ \ aux_\psi(\bar{x}'), \bar{x}' \neq \textit{null}.$$

$$aux_\psi(\bar{x}') \leftarrow Q(\bar{x}', \overline{\textit{null}}), \ \textit{not } Q_-(\bar{x}', \overline{\textit{null}}, \mathbf{f_a}), \bar{x}' \neq \textit{null}.$$

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

$$aux_\psi(\bar{x}') \leftarrow Q_-(\bar{x}', \bar{y}, \mathbf{t}^\star), \ \textit{not } Q_-(\bar{x}', \bar{y}, \mathbf{f_a}), \bar{x}' \neq \textit{null}, y_i \neq \textit{null}.$$

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

$$R_-(\bar{x}, \mathbf{f}^\star) \leftarrow \textit{dom}(\bar{x}), \ \textit{not } R(\bar{x}).$$

$$R_-(\bar{x}, \mathbf{f}^\star) \leftarrow R_-(\bar{x}, \mathbf{f_a}).$$

$$R_-(\bar{x}, \mathbf{t}^\star) \leftarrow R(\bar{x}).$$

$$R_-(\bar{x}, \mathbf{t}^\star) \leftarrow R_-(\bar{x}, \mathbf{t_a}).$$

9. For each predicate $R \in \mathcal{R}(\mathtt{P})$, the interpretation rule:

$$R_-(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow R_-(\bar{x}, \mathbf{t}^\star), \ \textit{not } R_-(\bar{x}, \mathbf{f_a}).$$

10. For each predicate $R \in \mathcal{R}(\mathcal{N}(\mathtt{P}))$, the program denial constraint:

$$\leftarrow R\_(\bar{x}, \mathbf{t_a}), R\_(\bar{x}, \mathbf{f_a}). \qquad \square$$

The facts in this new solution program are those in the instance of $\mathtt{P}$ and, those in one instances of each neighbor $\mathtt{P'}$ of $\mathtt{P}$ (not necessarily equal to $D(\mathtt{P'})$). In Definition 7.19 the instance offered to $\mathtt{P}$ by neighbor $\mathtt{P'}$ will be a solutions of peer $\mathtt{P'}$ (under the same semantics).

Definition 7.15 The *P2P instance* associated to a stable model $\mathcal{M}$ of program $\Pi^{\mathtt{II}}(\mathtt{P}, \mathcal{S}, \mathfrak{P})$ is $D_{\mathcal{M}} = \{R(\bar{a}) \mid R(\bar{a}, \mathbf{t^{\star\star}}) \in \mathcal{M} \text{ and } R \in \mathcal{R}(\mathcal{N}(\mathtt{P}))\}$. $\qquad \square$

Notice that if a peer $\mathtt{P}$ has no DECs, then $\Pi^{\mathtt{II}}(\mathtt{P}, \mathcal{S}, \mathfrak{P})$ is reduced to $\Pi(D(\mathtt{P}), IC(\mathtt{P}))$ (see Definition 5.6). This implies that the *P2P instances* associated to the stable models of $\Pi^{\mathtt{II}}(\mathtt{P}, \mathcal{S}, \mathfrak{P})$ correspond to the repairs of the database in peer $\mathtt{P}$ with respect to its ICs.

Proposition 7.2 Given a peer $\mathtt{P}$ in a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$, if there is no peer $\mathtt{Pi}$ such that $\Sigma(\mathtt{P}, \mathtt{Pi}) \in \Sigma$ and $\Sigma(\mathtt{P}, \mathtt{Pi}) \neq \emptyset$, then the set of P2P instances associated to $\Pi^{\mathtt{II}}(\mathtt{P}, \mathcal{S}, \mathfrak{P})$ is the same as the set of repairs obtained from $\Pi(D(\mathtt{P}), IC(\mathtt{P}))$ (see Definition 5.6). $\qquad \square$

The following is a recursive definition of a global solution for $\mathtt{P}$ under semantics $\mathtt{II}$ for a peer that uses the solution program with $\mathcal{S}$ being the solutions, under the same semantics, of the neighboring peers.

Definition 7.16 (transitive case, semantics $\mathtt{II}$) Given a peer $\mathtt{P}$ in a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$, where the graph $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\mathtt{P})]$ is acyclic and $\bigcup_{\mathtt{Pi} \in \mathcal{N}(\mathtt{P})}(\Sigma(\mathtt{Pi}) \cup IC(\mathtt{Pi}))$ is RIC-acyclic; an instance D over $\mathcal{R}(\mathcal{N}(\mathtt{P}))$ is a *global solution under semantics* $\mathtt{II}$ for $\mathtt{P}$ if there exist solutions $s_{\mathtt{P1}}$, ..., and $s_{\mathtt{Pn}}$ (also under

semantics II) for P1, ..., and Pn in $\mathcal{N}(\mathtt{P})$; and a stable model $\mathcal{M}$ of $\Pi(\mathtt{P}, \{s_{\mathtt{P1}}, \ldots,$ $s_{\mathtt{Pn}}\}, \mathfrak{P})$, such that $D = D_{\mathcal{M}}$. $S_G^{\mathtt{II}}(\mathtt{P})$ is the set of global solutions for peer P under semantics II.

An instance s is a *solution under semantics* II for peer P if there is a global solution D for P such that $s = D|\mathtt{P}$. $S(\mathtt{P})$ is the set of solutions for P. As in the local case, P's *peer consistent answers* (PCA) are those answers that can be retrieved from *every* solution for P. □

For a peer P we need one program $\Pi(\mathtt{P}, \{s_{\mathtt{P1}}, \ldots, s_{\mathtt{Pn}}\}, \mathfrak{P})$, for each combination $\{s_{\mathtt{P1}},$ $\ldots, s_{\mathtt{Pn}}\}$ of solutions of the neighboring peers. The solutions of P are the union of the solutions obtained from each program.

Example 7.20 If a peer P has two neighbors, P1 and P2, such that $S(\mathtt{P1})$ and $S(\mathtt{P2})$ are of cardinality two and four respectively, there will be eight possible combinations of solutions, and therefore eight programs $\Pi^{\mathtt{II}}$. The solutions of P will be the union of the solutions obtained from each of the eight programs. □

Since Definition 7.16 is recursive, it can only be used for a peer P for which the graph $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\mathtt{P})]$ is acyclic. If the graph is acyclic, the recursion will reach the peers in the leaves of $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\mathtt{P})]$. Since those peers will have no outgoing edges, their solutions can be obtained without the need for solutions of other peers. In this way the recursive definition will always terminate.

Example 7.21 Definition 7.16 does not define the solutions for peer P1 for the peer data exchange system shown in Figure 7.9, since there is a cycle in $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\mathtt{P1})]$ between peers P2, P3 and P4. However, for peer P5, $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\mathtt{P5})]$ is acyclic and therefore we would only need to check if $\bigcup_{\mathtt{Pi} \in \mathcal{N}(\mathtt{P5})}(\Sigma(\mathtt{Pi}) \cup IC(\mathtt{Pi}))$ is RIC-acyclic in order to use semantics given by Definition 7.16 for peer P5. □
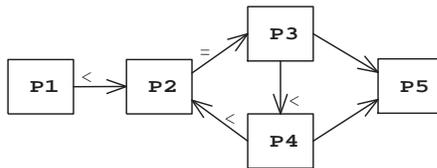
Figure 7.9: Cyclic accessibility graph of Example 7.21

Computation of PCA under Semantics II

Like for semantics I, each peer should be capable of answering queries from users (user-queries) and from peers (peer-queries). Answers to user-queries are PCAs and answers to peer-queries are solutions.

The following procedure is a naive implementation of semantics II: when a user-query is posed to a peer P, it will need to compute its solutions. In order to do this, it needs the solutions of its neighboring peers. They can be obtained by peer P by sending a peer-query to its neighbors. Each neighboring peer will need the solutions of its neighbors to calculate its solutions. Since the accessibility graph is acyclic, this recursion will, at some point, reach peers that have no DECs and therefore, by Proposition 7.2, the solutions of this peers can be calculated by using only local information. The solutions obtained are returned to the peer Pj that sent the peer-query. Peer Pj takes the solutions sent by all its neighbors and construct a program $\Pi^{\text{II}}(\text{Pj}, \mathcal{S}_k, \mathfrak{P})$ for every combination $\mathcal{S}_k$ of solutions of the neighbors (using one solutions from each peer). The solutions of Pj are the union of the solutions obtained from each program $\Pi^{\text{II}}(\text{Pj}, \mathcal{S}_k, \mathfrak{P})$. The solutions are sent again to the peer that posed the peer-query to Pj. This process continues until P is reached. Finally, P computes its own solutions and with them the PCAs to the user-query.

Example 7.22 (example 7.11 and 7.18 continued) $\mathcal{G}_A(\mathfrak{P})$ has no cycle and the set of DECs and IC is RIC-acyclic, as a consequence, the solutions are defined for all peers. If a user-query is posed to peer P1, P1 will need to send a peer-query to

`P2` to get its solutions. In turn, `P2` will need send a peer-query to `P3` to get its solutions. Since `P3` has no DECs, it can obtain its solution by running the program $\Pi^{II}(\text{P3}, \emptyset, \mathfrak{P}) = \Pi(D(\text{P3}), IC(\text{P3}))$:

$dom(c).$ $dom(d).$ $\dots$

$R^3(c, 4).$

$R^3_{-}(x, y, \mathbf{t^\star}) \leftarrow R^3_{-}(x, y, \mathbf{t_a}).$

$R^3_{-}(x, y, \mathbf{t^\star}) \leftarrow R^3(x, y).$

$R^3_{-}(x, y, \mathbf{f^\star}) \leftarrow R^3_{-}(x, y, \mathbf{f_a}).$

$R^3_{-}(x, y, \mathbf{f^\star}) \leftarrow dom(x), dom(y),\ not\ R^3(x, y).$

$R^3_{-}(x, y, \mathbf{t^{\star\star}}) \leftarrow R^3_{-}(x, y, \mathbf{t^\star}),\ not\ R^3_{-}(x, y, \mathbf{f_a}).$

$\leftarrow R^3_{-}(x, y, \mathbf{t_a}), R^3_{-}(x, y, \mathbf{f_a}).$

The only model of this program gives the solution $s_{\text{P3}} = \{R^3(c, 4)\} = D(\text{P3})$. Peer `P3` sends this solution to peer `P2`. Now, the solutions of `P2` are computed using the unique solution of `P3`. $\Pi^{II}(\text{P2}, \{s_{\text{P3}}\}, \mathfrak{P})$ is:

$dom(c).$ $dom(d).$ $\dots$

$R^2(c, 4).$ $R^2(d, 5).$ $R^3(c, 4).$

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

$\left.\begin{array}{l} R^2_{-}(x, y, \mathbf{t^\star}) \leftarrow R^2_{-}(x, y, \mathbf{t_a}). \\[4pt] R^2_{-}(x, y, \mathbf{t^\star}) \leftarrow R^2(x, y). \\[4pt] R^2_{-}(x, y, \mathbf{f^\star}) \leftarrow R^2_{-}(x, y, \mathbf{f_a}). \\[4pt] R^2_{-}(x, y, \mathbf{f^\star}) \leftarrow dom(x), dom(y),\ not\ R^2(x, y). \\[4pt] R^2_{-}(x, y, \mathbf{t^{\star\star}}) \leftarrow R^2_{-}(x, y, \mathbf{t^\star}),\ not\ R^2_{-}(x, y, \mathbf{f_a}). \\[4pt] \leftarrow R^2_{-}(x, y, \mathbf{t_a}), R^2_{-}(x, y, \mathbf{f_a}). \end{array}\right\}$ (Similarly for S^2 and R^3)

The models of this program give $S_G(\text{P2}) = \{\{R^2(d, 5), R^2(c, 4)\}, \{R^2(d, 5), R^3(c, 4)\}\}$. As a consequence, the solutions are $s_{\text{P2}_1} = \{R^2(d, 5)\}$ and $s_{\text{P2}_3} = \{R^2(c, 4), R^2(d, 5)\}$. The solutions to `P2` are separately sent to `P1` and its solutions are computed with the

two programs $\Pi^{\text{II}}(\texttt{P1}, \{s_{\texttt{P2}_1}\}, \mathfrak{P})$ and $\Pi^{\text{II}}(\texttt{P1}, \{s_{\texttt{P2}_2}\}, \mathfrak{P})$. The first program leads to global solution $\{\{R^1(a,2),\ R^2(c,4),\ R^2(d,5),\ R^1(c,4),\ R^1(d,5)\}$, and the second one to $\{R^1(a,2),\ R^2(d,5),\ R^3(c,4),\ R^1(d,5)\}$. Therefore, $S_G^{\text{II}}(\texttt{P1}) = \{\{R^1(a,2),\ R^2(c,4),\ R^2(d,5),\ R^1(c,4),\ R^1(d,5)\}, \{R^1(a,2),\ R^2(d,5),\ R^3(c,4),\ R^1(d,5)\}\}$ and $S^{\text{II}}(\texttt{P1}) = \{\{R^1(a,2),\ R^1(c,4),\ R^1(d,5)\},\ \{R^1(a,2),\ R^1(d,5)\}\}$. $\qquad\square$

In some cases, such as Example 7.22, the same solutions would have been obtained by putting together all the programs run in each peer. In fact, this unified program would coincide with the program for semantics I. In Section 7.4 we study conditions under which semantics I and II coincide.

Since semantics II requires the accessibility graph to be acyclic (no cycles through trust relationships), a unique ID has to be created for each user-query, and kept in all the peer-queries that originate from it. In this way, if a cycle through trust relationships is found, an error should be given as an answer to the user-query.

The implementation of the semantics can be optimized in several ways. For example we can calculate only the portions of the solutions that are relevant to the user-query and DECs.

7.3.3 Solutions under Semantics III

The third semantics consists of modifying the local specification in such a way that it considers the data in peer P; and in the intersection of all its solutions for each for each neighboring peer; and peer P's DECs and ICs. Under this semantics, the transitive information is gathered in the intersection of the solutions of the neighboring peers.

Example 7.23 (example 7.2, 7.11 and 7.18 continued) Under semantics III, the solutions for peer P3 will only be affected by its own data, the intersection of the solutions of peer P2, the DECs between them and the ICs of P3. Therefore, in order

to compute the solutions for peer P3, the solutions of P2 are needed. In its turn, the solutions for P2 will be affected by its own data, the intersection of the solutions of peer P3, the DECs between them and the ICs of P2. Finally, since P3 has no DECs nor ICs, its solution is D(P3). $\square$

Definition 7.17 Consider a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$, a peer P $\in \mathcal{P}$ and the set $\mathcal{S} = \{s_{P1}, \ldots, s_{Pn}\}$, where $\mathcal{N}(P) = \{P, P1, \ldots, Pn\}$ and s_{Pj} is a database with the same schema as Pj. The *solution program for semantic* III, $\Pi^{III}(P, \mathcal{S}, \mathfrak{P})$ is:

1. $dom(x)$, for every $x \in (\mathcal{U} \smallsetminus \{null\})$

2. $R(\bar{a})$, for each atom $R(\bar{a}) \in \bar{D}(P)$.

3. $R(\bar{a})$, for each $R(\bar{a}) \in s$ with $s \in \mathcal{S}$.

4. Same as rules 4. to 10. of $\Pi^{II}(P, \mathcal{S}, \mathfrak{P})$ in Definition 7.14. $\square$

The facts in this new solution program are those in the instance of P and, those in one instances of each neighbor P' of P (not necessarily equal to D(P')). In Definition 7.19 the instance offered to P by neighbor P' will be the intersections of the solutions of peer P' (under the same semantics). If there are cycles through constraints, i.e., $\Sigma \cup IC$ is RIC-cyclic, the solutions given by the semantics can be counterintuitive as shown in Example 5.12.

Definition 7.18 The P2P instance associated to a stable model $\mathcal{M}$ of program $\Pi^{III}(P, \mathcal{S}, \mathfrak{P})$ is $D_{\mathcal{M}} = \{R(\bar{a}) \mid R(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}$ and $R \in \mathcal{R}(\mathcal{N}(P))\}$. $\square$

Notice that if a peer P has no DECs, then $\Pi^{III}(P, \mathcal{S}, \mathfrak{P})$ is reduced to $\Pi(D(P), IC(P))$ (see Definition 5.6). This implies that the P2P instance associated to the stable models of $\Pi^{III}(P, \mathcal{S}, \mathfrak{P})$ correspond to the repairs of the database in peer P with respect to its ICs.

Proposition 7.3 Given a peer P in a P2P system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$, if there is no peer Pi such that $\Sigma(\texttt{P},\texttt{Pi}) \in \Sigma$ and $\Sigma(\texttt{P},\texttt{Pi}) \neq$, then the set of P2P instances associated to $\Pi^{\texttt{III}}(\texttt{P}, \mathcal{S}, \mathfrak{P})$ is the same as the set of repairs obtained from $\Pi(D(\texttt{P}), IC(\texttt{P}))$ (see Definition 5.6). □

The following is a recursive definition of a solution under semantics III for a peer that uses the solution program for semantics III with $\mathcal{S}$ being the intersection of the solutions, under the same semantics, of the neighboring peers.

Definition 7.19 (transitive case, semantics III) Given a peer P in a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$, where the graph $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\texttt{P})]$ is acyclic and $\bigcup_{\texttt{Pi} \in \mathcal{N}(\texttt{P})}(\Sigma(\texttt{Pi}) \cup IC(\texttt{Pi}))$ is RIC-acyclic; an instance D over $\mathcal{R}(\mathcal{N}(\texttt{P}))$ is a *global solution under semantics* III for P if there exist a stable model $\mathcal{M}$ of $\Pi^{\texttt{III}}(\texttt{P}, \{s_{\texttt{P1}}, \ldots, s_{\texttt{Pn}}\}, \mathfrak{P})$, such that $D = D_{\mathcal{M}}$ and $s_{\texttt{Pi}} = \bigcap S^{\texttt{III}}(\texttt{Pi})$ for $i = 1, \ldots, n$. $S_G^{\texttt{III}}(\texttt{P})$ is the set of global solutions for peer P under semantics III.

An instance s is a *solution under semantics* III for peer P if there is a global solution D for P such that $s = D|\texttt{P}$. $S(\texttt{P})$ is the set of solutions for P. As in the local case, P's *peer consistent answers* (PCA) are those answers that can be retrieved from *every* solution for P. □

Note that since this is a recursive definition, it can only be used for peers P such that $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\texttt{P})]$ is acyclic, i.e. without cycles through trust relationships. If the graph is acyclic, the recursion will terminate when a peer with no outgoing edges in $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\texttt{P})]$ is reached, since its solution can be computed without the need of the intersection of solutions of other peers.

Computation of PCAs under Semantics III

In order to compute solutions under semantics I and II each peer had to be capable of answering queries from users (user-queries) and from peers (peer-queries). Under semantics III we only have one type of queries, this is, the peer answers in the same way user and peer queries. In both cases it returns peer consistent answers. As it will be explained later, the only difference between queries between peers and between a user and a peer, is that in a naive implementation, the queries between peers will request whole tables. On the other hand, a query between a peer and a user can be more specific.

The following procedure is a naive implementation of semantics III: when a query is posed to a peer P, it will need to compute its solutions. In order to do this, it needs the intersection of the solutions of its neighboring peers. They can be obtained by posing several query to its neighbors requesting all the data in each table that is needed to check the satisfaction of the DECs. The neighbor might have other neighbors that it will need to query too. Since the accessibility graph is acyclic, this recursion will, at some point, reach peers that have no DECs and therefore, by Proposition 7.2, the PCAs of this peers can be calculated by using only local information. The PCAs obtained are returned to the peer that sent the query. The PCAs are used to construct a program $\Pi^{\mathrm{III}}(\mathtt{Pj}, \mathcal{S}, \mathfrak{P})$. The PCAs are sent again to the peer that posed the query to Pj. This process continues until P is reached. Finally, P computes its own solutions and with them the PCAs to the query that originated the process.

Since semantics III requires that the accessibility graph is acyclic, for each query a unique ID has to be created and kept in all the queries that originate from it. In this way, if a cycle is found, an error should be given as an answer to the all the related queries.

Example 7.24 (example 7.11 and 7.22 continued) If a user poses query $Q_0 = R^1(x, y)$ we will need to compute the solutions of peer P1 in order to compute the PCAs. To get the solutions of P1 we need the intersection of the solutions of peer P2. The intersection can be obtained from the PCAs of the queries $Q_1 : R^2(x, y)$ and $Q_2 : S^2(x, y)$ posed to P2.

In order to answer these queries, peer P2 needs the intersection of the solutions of peer P3 which corresponds to the PCAs of query $Q_3 : R^3(x, y)$. Since P3 has no neighboring peers, $\Pi^{III}(\text{P3}, \emptyset, \mathfrak{P}) = \Pi(D(\text{P}), IC(\text{P}))$ by Proposition 7.3. The only model of this program gives the solution $s_{\text{P3}} = \{R^3(c, 4)\} = D(\text{P3})$. Now since P3 has a unique solution, the PCAs to query Q_3 sent to peer P2 is $\{(c, 4)\}$.

Peer P2 now knows that the intersection of the solutions of P3 is $\{R^3(c, 4)\}$. The model of program $\Pi^{III}(\text{P2}, \{\{R^3(c, 4)\}\}, \mathfrak{P})$ gives $S_G(\text{P2}) = \{\{R^2(d, 5), R^2(c, 4)\}, \{R^2(d, 5), R^3(c, 4)\}\}$. As a consequence, the solutions for peer P2 are $S^{III}(\text{P2}) = \{\{R^2(d, 5)\}, \{R^2(c, 4), R^2(d, 5)\}\}$. The PCAs to queries Q_1 and Q_2 sent to peer P1 are $\{(d, 5)\}$ and $\emptyset$ respectively. With this information P1 now knows that the intersection of the solutions of P2 is $\{R^2(d, 5)\}$. Now, Π^{III} (P1, $\{\{R^2(d, 5)\}\}, \mathfrak{P})$:

$dom(c).\quad dom(d).\quad \ldots \quad R^1(a, 2)\quad R^2(d, 5).$

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

$R^1_\_(x, y, \mathbf{t^\star}) \leftarrow R^1_\_(x, y, \mathbf{t_a}).$

$R^1_\_(x, y, \mathbf{t^\star}) \leftarrow R^1(x, y).$

$R^1_\_(x, y, \mathbf{f^\star}) \leftarrow R^1_\_(x, y, \mathbf{f_a}).$

$R^1_\_(x, y, \mathbf{f^\star}) \leftarrow dom(x), dom(y),\ not\ R^1(x, y).$

$R^1_\_(x, y, \mathbf{t^{\star\star}}) \leftarrow R^1_\_(x, y, \mathbf{t^\star}),\ not\ R^1_\_(x, y, \mathbf{f_a}).$

$\leftarrow R^1_\_(x, y, \mathbf{t_a}), R^1_\_(x, y, \mathbf{f_a}).$

(Similarly for R^2)

The program leads to two global solutions: $S_G^{III} = \{\{R^1(a, 2)\}, \{R^1(a, 2), R^2(d, 5),$

$R^1(d, 5)\}\}$. Therefore, $S^{\text{III}}(\text{P1}) = \{\{R^1(a, 2)\}, \{R^1(a, 2), R^1(d, 5)\}\}$. From the solutions we get that the PCA to Q_0 is $\{(a, 2)\}$. □

The implementation of the semantics can be optimized in several ways. For example we can make more specific queries to the neighboring peers in such a way that only the relevant portion of the intersection of the solutions is retrieved.

Example 7.25 (example 7.24 continued) In order to get the PCAs of peer P1 we do not need the whole intersection of the solutions of peer P2. In fact, the DEC $\Sigma(\text{P1}, \text{P2}) = \{\forall xy(R^2(x, y) \to R^1(x, y))\}$ shows that we only need relation R^2 and therefore query $Q_1 : R^2(x, y)$ was enough to check the satisfaction of the DEC. □

Example 7.26 For a DEC $\Sigma(\text{P1}, \text{P3}) = \{R^1(x, y) \to R^2(x, y, a)\}$, peer P1 needs only the data in table R^2 to check the satisfaction of the DEC. In the naive algorithm we explained above we would request the whole table (and any other table in P2. An alternative would be to obtain only the information that will be needed to check the constraint. In this case, peer P1 could send to P2 the query $Q_1(x, y) : R^2(x, y, a)$. Using the answers to Q_1 instead of the intersection of all the solutions would give the same PCAs to any query posed to P1.

Now, for a DEC $\Sigma(\text{P1}, \text{P3}) = \{R^1(x, y) \to R^3(x, z)\}$, it is not the same to pose query $Q_2(x) : \exists z R^3(x, z)$ to P3 than using the intersection of the databases. For example, if $S(\text{P3}) = \{\{R^3(a, b)\}, \{R^3(a, c)\}\}$, the intersection of the solutions would be empty and the answer to query Q_2 would be $\{(a)\}$. □

If the DECs were defined in terms of queries, then it would be possible to use PCA of specific queries instead of the intersection of the solutions.
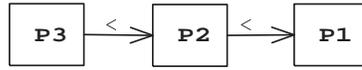
Figure 7.10: Accessibility graph of Example 7.27

7.4 Comparison of Semantics for the Transitive Case

The different semantics can be useful depending on the characteristics and architecture of the PDMs.

Semantics I is seeing the P2P system as a database containing all the data in the system subject to a set of ICs that contain the ICs of all the peers and the DECs. On the other hand, Semantics III is in the other end, where we see each peer as an individual database that interacts with other peers only through PCAs. Semantics II would be somewhere in the middle of Semantics I and III, since the computation is distributed over the network, but the data being send between peers are solutions to the peers instead of PCAs.

Example 7.27 Consider a query $Q : Citizen^3(x)$ posed to peer P3 in the P2P data exchange system $\mathfrak{P}$:[6]

P1: $\mathcal{R}(\text{P1}) = \{Citizen(\cdot)\}, \quad D(\text{P1}) = \{Citizen^1(125)\},$

P2: $\mathcal{R}(\text{P2}) = \{Man^2(\cdot), Woman^2(\cdot)\}, \quad D(\text{P2}) = \{Man^2(562), Woman^2(167)\},$

P3: $\mathcal{R}(\text{P3}) = \{Citizen^3(\cdot)\}, \quad D(\text{P3}) = \{\},$

$\Sigma(\text{P2}, \text{P1}) = \{\forall x(Citizen^1(x) \rightarrow (Man^2(x) \vee Woman^2(x)))\},$

$\Sigma(\text{P3}, \text{P2}) = \{ \forall x(Man^2(x) \rightarrow Citizen^3(x)) , \forall x(Woman^2(x) \rightarrow Citizen^3(x))\},$

$trust = \{ (\text{P2}, less, \text{P1}), \ (\text{P3}, less, \text{P2}) \}.$

Figure 7.10 shows the accessibility graph of this system.

The global solutions obtained from $\Pi^{\mathtt{I}}(\text{P3}, \mathfrak{P})$ are $S_G^{\mathtt{I}}(\text{P3}) = \{\{Citizen^1(125),$ $Man^2(562), Woman^2(167), Man^2(125), Citizen^3(562), Citizen^3(167), Citizen^3(125)\},$

---

[6]The same example is used in [Franconi *et al.*, 2004b]

$\{Citizen^1(125),\ Man^2(562),\ Woman^2(167),\ Woman^2(125),\ Citizen^3(562),\ Citizen^3(167),$
$Citizen^3(125)\}\}$. Therefore, there is only one solution for P3: $S^{\mathrm{I}}(\mathrm{P3}) = \{\{Citizen^3(562),$
$Citizen^3(167),\ Citizen^3(125)\}\}$. The PCAs under semantics I are $\{562, 167, 125\}$.

Now, under semantics II, in order to compute the solution of P3, we need the solutions of P2, which need the solutions of P1. Since P1 has no DECs, its solutions can be computed from $\Pi^{\mathrm{II}}(\mathrm{P1}, \emptyset, \mathfrak{P})$. Since there are no ICs in P1, the solution obtained from the program coincides with $D\mathrm{P1}$, $S^{\mathrm{II}}(\mathrm{P1}) = \{\{Citizen^1(125)\}\}$. Using this solution we can compute the solutions of P2 with the program $\Pi^{\mathrm{II}}(\mathrm{P2}, \{\{Citizen^1(125)\}\}, \mathfrak{P})$. The solutions obtained from it are $S^{\mathrm{II}}(\mathrm{P2}) = \{\{Man^2(562),\ Woman^2(167),\ Man^2(125)\},$ $\{Man^2(562),\ Woman^2(167),\ Woman^2(125)\}\}$. Using these solutions we can compute the solutions to peer P3 from programs $\Pi^{\mathrm{II}}(\mathrm{P3}, \{\{Man^2(562),\ Woman^2(167),$ $Man^2(125)\}\}, \mathfrak{P})$ and $\Pi^{\mathrm{II}}(\mathrm{P3}, \{\{Man^2(562),\ Woman^2(167),\ Woman^2(125)\}\}, \mathfrak{P})$. In this case, the same solution for peer P3 is obtained from both programs and $S^{\mathrm{III}} = \{\{Citizen^3(567),\ Citizen^3(167),\ Citizen^3(125)\}\}$. The PCAs under semantics II are $\{562, 167, 125\}$ and coincide with those obtained under semantics I.

Finally, under semantics III, in order to compute the solution of P3, peer P3 needs the intersection of the solutions of P2, which needs the intersection of the solutions of P1. Since P1 has no DECs, its solutions can be computed from $\Pi^{\mathrm{III}}(\mathrm{P1}, \emptyset, \mathfrak{P})$. Since there are no ICs in P1, there is only one solution and it coincides with $D\mathrm{P1}$, $S^{\mathrm{II}}(\mathrm{P1}) = \{\{Citizen^1(125)\}\}$. Since there is unique solution, the intersection is the same as the solution. This intersection is sent to peer P2 and is used to compute the solutions of P2 with the program $\Pi^{\mathrm{II}}(\mathrm{P2}, \{\{Citizen^1(125)\}\}, \mathfrak{P})$. The solutions obtained from it are $S^{\mathrm{II}}(\mathrm{P2}) = \{\{Man^2(562),\ Woman^2(167),\ Man^2(125)\},$ $\{Man^2(562),\ Woman^2(167),\ Woman^2(125)\}\}$. The intersection of the solutions is $s_{\mathrm{P2}} = \{Man^2(562),\ Woman^2(167)\}$ and is send to peer P1. Using the intersection of the solutions we can compute the solutions to peer P3 from program $\Pi^{\mathrm{II}}(\mathrm{P3}, \{\{Man^2$

(562), $Woman^2(167)\}\}, \mathfrak{P})$. A unique solution is obtained from the program: $S^{\text{III}}(\text{P3})$ $= \{\{Citizen^3(562), \ Citizen^3(167)\}\}$. The PCAs under semantics III are $\{562, 167\}$.

The PCAs for semantics I and II coincide in this case. The PCAs under semantics III do not coincide with the other ones since we only send between peers the information that we are certain of. $\square$

In Examples 7.22 and 7.27 the solutions under semantics I coincide with those under semantics II, but this is not always the case as the following example shows.

Example 7.28 Consider a P2P data exchange system $\mathfrak{P}$:

P1: $\mathcal{R}(\text{P1}) = \{R^1(\cdot, \cdot)\}, \ D(\text{P1}) = \{\},$

P2: $\mathcal{R}(\text{P2}) = \{S^2(\cdot, \cdot), T^2(\cdot, \cdot)\}, \ D(\text{P2}) = \{S^2(a, d), R^2(a, d)\},$

$\Sigma(\text{P1}, \text{P2}) = \{ \ \forall xy(S^2(x, y) \rightarrow R^1(x, y)) \}$

$IC(\text{P2}) = \{ \ \forall xy(T^2(x, y) \rightarrow S^2(x, y)) \}$

Figure 7.11 shows the accessibility graph for $\mathfrak{P}$.
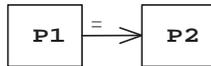
Figure 7.11: Accessibility graph of Example 7.28

For semantics I, there are two global solutions for P1, $S_G^{\text{I}}(\text{P1}) = \{\{S^2(a, d), T^2(a, d), R^1(a, d)\}, \emptyset\}$ and, therefore $S^{\text{I}} = \{\{R^1(a, d)\}, \emptyset\}$.

On the other hand, for semantics II we need first to compute the solutions of peer P2. Since they have no DECs and the only IC is satisfied by P2's instance, $S_G^{\text{II}}(\text{P2}) = S^{\text{I}}(\text{P2}) = \{\{S^2(a, d), T^2(a, d)\}\}$. The set of global solutions for P2 under semantics II is $S_G^{\text{II}}(\text{P1}) = \{\{S^2(a, d), T^2(a, d), R^1(a, d)\}\}$, and therefore $S^{\text{I}} = \{\{R^1(a, d)\}\}$.

Under semantics I, peer P1 has two solutions and under semantics II it has only one. Therefore, semantics I and II do not necessarily coincide. $\square$

The following propositions state that if there are only *less* relationships in *trust* and there are no cycles in the DECs, then solution semantics I and II coincide.

Proposition 7.4 Given a peer P and a P2P system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$ such that P $\in \mathcal{P}$, $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\text{P})]$ has no cycles that include P and *trust* has only relations of type *less* for peers in AC(P), it holds

- For every stable model $\mathcal{M}$ of $\Pi^{\text{I}}(\text{P}, \mathfrak{P})$ there exists a set $\mathcal{S} = \{s_{\text{P1}}, \ldots, s_{\text{Pn}}\}$ for $\mathcal{N}(\text{P}) = \{\text{P1}, \ldots, \text{Pn}\}$, and a model $\mathcal{M}'$ of $\Pi^{\text{II}}(\text{P}, \mathcal{S}, \mathfrak{P})$, such that each s_{Pi} is a solution under semantics II for Pi and $D_{\mathcal{M}} | \mathcal{R}(\mathcal{N}(\text{P})) = D_{\mathcal{M}'}$.

- For every stable model $\mathcal{M}$ of $\Pi^{\text{II}}(\text{P}, \mathcal{S}, \mathfrak{P})$, where $\mathcal{S} = \{s_{\text{P1}}, \ldots, s_{\text{Pn}}\}$, $\mathcal{N}(\text{P}) = \{\text{P1}, \ldots, \text{Pn}\}$ and s_{Pi} is a solution under semantics II for Pi, there exists a model $\mathcal{M}'$ of $\Pi^{\text{I}}(\text{P}, \mathfrak{P})$, such that $D_{\mathcal{M}'} | \mathcal{R}(\mathcal{N}(\text{P})) = D_{\mathcal{M}}$.

Proof: Program $\Pi^{\text{I}}(\text{P}, \mathfrak{P})$ can be replaced by a group of programs of the type $\Pi^{\text{II}}(\text{P}, \mathcal{S}, \mathfrak{P})$, by using the splitting theorem [Lifschitz and Turner, 1994]. The theorem can be applied because:

- There are no cycles through the DECs of the peers in $\mathcal{AC}(\text{P})$

- The repairs performed to restore the consistency of any DEC or IC of a peer P will only affect the data in itself. This is a consequence of all trust relationships being of type *less*. □

A direct consequence of Proposition 7.4 is the following corollary.

Corollary 7.1 For a P2P data exchange system $\mathfrak{P} = \langle \mathcal{P}, \Sigma, IC, trust \rangle$ such that the graph $\mathcal{G}_A(\mathfrak{P})[\mathcal{AC}(\text{P})]$ has no cycles that include P, and *trust* has only relations of type *less*, it holds that $S^{\text{I}}(\text{P}) = S^{\text{II}}(\text{P})$, i.e., the solutions provided by both semantics are the same. □

Intuitively this can be explained as follows: since $\mathcal{AC}(\mathtt{P})$ is acyclic we can see the graph as a tree with P as the root. Since the trust relationships are always *less*, the repairs will only have effects under Semantics I and II going from leaf to root. This implies that we can separate the solution program under semantics I into several layers. On the other hand, if we had trust relationships of the type *same*, the modifications could also move in the direction root to leaf. In the case of semantic II, the impact of those modifications would be contained in the neighborhood, but in the case of semantics I it could go as far as the peer in the leaf. Therefore, for RDECs and UDECs, if there is a trust relationship *same*, semantics I and II do not necessarily coincide.

The following example shows that it is also possible that the three semantics coincide. This can happen if there are no cycles in the accessibility graph and there is only one way to solve inconsistencies with respect to DECs and ICs (no disjunctions in the head of the rules that restore consistency).

Example 7.29 Consider the P2P system $\mathfrak{P}$:

$$\mathcal{R}(\mathtt{P1}) = \{R^1(\cdot,\cdot), S^1(\cdot,\cdot,\cdot)\}, \quad D(\mathtt{P1}) = \{S^1(a,b,c)\}$$

$$\mathcal{R}(\mathtt{P2}) = \{R^2(\cdot,\cdot), S^2(\cdot)\}, \quad D(\mathtt{P2}) = \{R^2(e,f), S^2(f)\}$$

$$\mathcal{R}(\mathtt{P3}) = \{R^3(\cdot,\cdot,\cdot)\}, \quad D(\mathtt{P2}) = \{R^3(b,e,e)\}$$

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

$$\Sigma(\mathtt{P2},\mathtt{P3}) = \{\forall xyz(R^3(x,y,z) \rightarrow R^2(x,y)), \forall xy(R^3(x,y,y) \rightarrow S^2(x))\},$$

$$trust = \{(\mathtt{P1},less,\mathtt{P2}), (\mathtt{P2},less,\mathtt{P3})\}.$$

In this case, semantics I, II and III coincide. The solutions are: $S(\mathtt{P1}) = \{\{S^1(a,b, c), R^1(e, f)\}\}$, $S(\mathtt{P2}) = \{\{R^2(e,f), R^2(b,e), S^2(f), S^2(b)\}\}$, $S(\mathtt{P3}) = \{\{R^3(b,e,e)\}\}$.
$\square$

In [Calvanese *et al.*, 2004b; Calvanese *et al.*, 2004a; Calvanese *et al.*, 2005], the semantics of a P2P system is given in terms of epistemic logic. They consider that each

peer is a data integration system and that there are no trust relationships between the peers. Since the DECs only modify the data of the peer that owns the DEC, they implicitly are assuming that the trust relationship between all peers is *less*. Another restriction of their setting is that every DEC in $\Sigma(\texttt{Pi},\texttt{Pj})$ can be written as $cq_j \rightarrow cq_i$, where cq_i and cq_j are conjunctive queries over peers $\texttt{Pi}$ and $\texttt{Pj}$ respectively. Therefore, their mappings can only add elements to another peer but not delete them.

The theory in epistemic logic is close in spirit to our semantics $\texttt{III}$, in the sense that only the data that is known to be true by a peer is used by the other peers to check the satisfaction of their DECs. However, in [Calvanese *et al.*, 2004b; Calvanese *et al.*, 2004a; Calvanese *et al.*, 2005] for a DEC $cq_j \rightarrow cq_i$ the data of peer $\texttt{Pj}$ is obtained with the conjunctive query cq_j, whereas in our case, we ask for intersection of the complete solutions. If cq_j has no existential quantifiers, the solutions for $\texttt{Pi}$ computed by using the answers to cq_j coincide with those obtained with the intersection of the solutions $\texttt{Pj}$.

Example 7.30 (example 7.29 continued) The PDMs $\mathfrak{P}$ can be adjusted to the settings in [Calvanese *et al.*, 2005] by transforming the database of every peer into a data integration system. Each peer $\texttt{P}$ is replaced by a DIS where:

- The global schema is $\mathcal{R}(\texttt{P})$.

- There is a unique source with a schema obtained from $\mathcal{R}(\texttt{P})$ by adding to the name each relation an s subscript. The source instance is $D(\texttt{P})$ where the subscript s is added to all the relations.

- For each $R \in \mathcal{R}(\texttt{P})$ add the GAV mapping $\forall \bar{x}(R_s(\bar{x}) \rightarrow R(\bar{x}))$.

Therefore, peer $\texttt{P1}$ has $\{R^1(\cdot,\cdot), S^1(\cdot,\cdot,\cdot)\}$ as global schema, $\{R_s^1(\cdot,\cdot), S_s^1(\cdot,\cdot,\cdot)\}$ as the source, the mapping $\{\forall xy(R_s^1(x,y) \rightarrow R^1(x,y)), \forall xyz(S_s^1(x,y,z) \rightarrow S^1(x,y,z))\}$

and the source instance $\{S_s^1(a,b,c)\}$. The DISs for the other peers are obtained analogously.

The theory in epistemic logic, as defined in [Calvanese *et al.*, 2005], is:

$$\mathbf{K_1}(\forall xy(R_s^1(x,y) \rightarrow R^1(x,y)))$$

$$\mathbf{K_1}(\forall xyz(S_s^1(x,y,z) \rightarrow S^1(x,y,z)))$$

$$\forall xy(\mathbf{K_2}(R^2(x,y) \wedge S^2(y)) \rightarrow \mathbf{K_1}(R^1(x,y)))$$

$$\text{Specification of } \texttt{P1}$$

$$\mathbf{K_2}(\forall xy(R_s^2(x,y) \rightarrow R^2(x,y)))$$

$$\mathbf{K_2}(\forall x(S_s^2(x) \rightarrow S^2(x)))$$

$$\forall xy(\mathbf{K_3}(R^3(x,y,y)) \rightarrow \mathbf{K_2}(S^2(x)))\}$$

$$\forall xyz(\mathbf{K_3}(R^3(x,y,z)) \rightarrow \mathbf{K_2}(R^2(x,y)))$$

$$\text{Specification of } \texttt{P2}$$

$$\mathbf{K_3}(\forall xyz(R_s^3(x,y,z) \rightarrow R^3(x,y,z)))$$

$$\} \text{ Specification of } \texttt{P3}$$

In this theory, $\mathbf{K_i}\phi$ can be interpreted as ϕ is known by peer $\texttt{Pi}$. A tuple $\bar{t}$ is a peer consistent answer to a query Q posed to peer $\texttt{Pi}$ if $\mathbf{K_i}Q(\bar{t})$ is a logical consequence of the theory above [Calvanese *et al.*, 2005]. What is known by the peers under the epistemic logic semantics coincides, in this case, with the PCAs under semantics $\texttt{I}$, $\texttt{II}$ and $\texttt{III}$. $\qquad\square$

Even though the semantics for peer consistent answers in [Calvanese *et al.*, 2005] is similar to our semantics $\texttt{III}$, theirs is able to deal with cycles in the accessibility graph.

Example 7.31 (example 7.29 and 7.30 continued) By adding DEC $\forall xyz(S^1(x,y,z) \rightarrow R^3(x,y,z))$ to peer $\texttt{P3}$ with trust relationship $(\texttt{P3}, less, \texttt{P1})$, the accessibility graph is now cyclic. Thus, semantics $\texttt{III}$ is not defined.

The epistemic logic semantics in [Calvanese *et al.*, 2005] can still be applied, since we can get the information of all the peers using a unique code to identify a query and

cut the cycle, and construct the epistemic logic theory in the peer that got the user query. This technique is the same as the one described in Section 7.3.1 for semantics I.

If peer P1 receives a user-query, it will first assign an ID to the query, e.g. $ID = A12$, and then send a peer-query with $ID = A12$ to peer P2 asking for its metadata (mappings, DECs and ICs) and data. Peer P2 will check if it has processed a peer-query with that ID. Since it has not, it will send a peer-query to P3. Peer P3 has not processed a query with that ID, therefore it will send a peer-query to P1. Peer P1 has processed this ID and therefore will not execute any peer-queries. Now peer P3 sends its information and data to P2, which in its turn, send the information and data to P1. Now peer 1 has all the needed data of the peers and the metadata needed to construct the epistemic logic theory:

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

$$\left.\begin{array}{l} \mathbf{K_2}(\forall xy(R_s^2(x,y) \rightarrow R^2(x,y))) \\ \mathbf{K_2}(\forall x(S_s^2(x) \rightarrow S^2(x))) \\ \forall xy(\mathbf{K_3}(R^3(x,y,y)) \rightarrow \mathbf{K_2}(S^2(x)))\} \\ \forall xyz(\mathbf{K_3}(R^3(x,y,z)) \rightarrow \mathbf{K_2}(R^2(x,y))) \end{array}\right\} \text{ Specification of P2}$$

$$\left.\begin{array}{l} \mathbf{K_3}(\forall xyz(R_s^3(x,y,z) \rightarrow R^3(x,y,z))) \\ \forall xyz(\mathbf{K_1}(S^1(x,y,z)) \rightarrow \mathbf{K_3}(R^3(x,y,z)))\} \end{array}\right\} \text{ Specification of P3}$$

By using this theory, and the data brought from the accessible peers, P1 is now ready to answer the user-query.

We cannot use the same technique for our semantics III, since in order to get $S(\text{P1})$ we need $S(\text{P2})$, which needs $S(\text{P3})$, which needs $S(\text{P1})$, etc. $\qquad\square$

In [Calvanese *et al.*, 2005], it is also note that it should be possible, in some cases, to use Datalog$^\neg$ to obtain the answers of a peer under their epistemic logic semantics.

Example 7.32 (example 7.31 continued) The queries to peer P1 can be answered by using the following Datalog program:

$$S^1(a, b, c). \qquad R^2(e, f).$$
$$S^2(f). \qquad R^3(b, e, e)$$
$$R^1(x, y, z) \leftarrow R^2(x, y), S^2(x, y).$$
$$R^2(x, y) \leftarrow R^3(x, y, z).$$
$$S^2(x) \leftarrow R^3(x, y, y).$$
$$R^3(x, y, z) \leftarrow S^1(x, y, z)$$

The answers to a query obtained using this program coincide with those obtained under the epistemic logic semantics. □

The algorithm in [Calvanese *et al.*, 2005] is distributed only in the sense that it has to traverse the network to get the mappings, DECs, ICs and data of the peers in order to construct the epistemic theory. The actual evaluation of the query is done by a single peer.

Another difference between the approach in [Calvanese *et al.*, 2005] and ours, is how they deal with inconsistencies with respect to ICs. If for a peer P, $D(\text{P})$ is inconsistent with respect to $IC(\text{P})$ the peer is discarded and not considered. The inconsistency can be detected when peer-queries are being sent to answer a user-query.

Example 7.33 (example 7.31 continued) If we add $IC(\text{P2}) = \{\forall xy(R^2(x, y) \rightarrow S^2(x))\}$, peer P2 would be inconsistent. Therefore, when computing the solutions for peer P1 under the semantics in [Calvanese *et al.*, 2005], the data of P2 is not considered. Since P3 was reachable from P1 through P2, its data will also not be considered. Thus, $S(\text{P}) = D(\text{P})$. □

Also, if data needed in a peer to restore consistency with respect to a DEC creates an inconsistency, then it will not be added to the peer. Our setting solves the inconsistencies in both cases without discarding whole peers and by restoring consistency for all DECs.

In [Calvanese *et al.*, 2005], it is shown, under the assumption that if the epistemic logic theory can be rewritten as a program in Datalog¬, then the data complexity of determining if a tuple is an answer is coNP-hard.[7] Also, under certain conditions over the mappings, the data complexity is in PTIME [Calvanese *et al.*, 2004a]. In our case, since we use the stable models semantics for disjunctive logic program, we have an upper bound of Π_2^P-hard, and clearly we can find situations with a Π_2^P lower bound. However, as we know from previous chapters, for certain classes of programs the complexity can be decreased. The identification of such cases is still a matter of current research.

[Franconi *et al.*, 2004b; Franconi *et al.*, 2004a] introduce a semantics that coincide with the epistemic logic used in [Calvanese *et al.*, 2004a]. They provide a distributed algorithm, were the data in the peers is updated by instruction of a *super peer* [Yang and Garcia-Molina, 2003]. When a query is possed to a peer, it can answer the query right away with the data store in the peer since the P2P system is already updated.

7.5 Conclusions

In this chapter, we have provided a framework for P2P exchange systems with trust relationships. In this setting, each peer solves its conflicts at query time, when it queries its own and other peers' databases.

For the local or direct case, in which the solution of a peer is only affected by the data in neighboring peers and its own DECs and ICS, we first provide a semantics

---

[7]It is not discussed in [Calvanese *et al.*, 2005] under which condition this assumptions apply.

for peers without *null* and explore logic program specification of it. Then, we extend the solution semantics to the case where peers may contain *null* and also use *null* to enforce the satisfaction of DECs and ICs. We provide a general logic program specification for this semantics when RDECs and UDECs are considered. This program can be used to retrieve the peer consistent answers.

For the transitive case, in which the solution of a peer is not only affected by its neighbors, but also by their interactions with other peers, we propose three alternative semantics. Under semantics I, the solution of a peer P is obtained by integrating the local specification of each of the peers that affect P. Under semantics II the solutions of a peer are defined using the solutions of the neighbors under the same semantics and its DECs and ICs. Finally, under semantics III the solutions are obtained with the data of the neighbors obtained as PCA, also under semantics III and its own DECs and ICs. We also identify some cases in which the semantics coincide.

The different semantics can be useful depending on the characteristics and architecture of the PDMs. For example, some P2P systems have so-called *super peers* [Yang and Garcia-Molina, 2003] that store information about the other peers and the data stored in them and also route queries. In this type of architecture, the super peer could get all the information it needs from the peers and implement the solution semantics I. In this setting, the super peer could also detect the presence of cycles and choose a way to break them in order to use semantics II or III. If, on the other hand, there are no super peers, we could use semantics II or III and if a cycle through trust relationships is detected, return an error. The cycles can be easily detected by adding a unique ID to identify the query and all its sub-queries. We are currently studying the use of the solution program as an immediate consequence operator, in order to extend semantics II and III to PDMs with cycles through trust relationships.

The choice of the semantics for the solution will also depend on the type and

granularity of data that is allowed to be sent between peers. If only queries and peer consistent answers can be exchanged between peers, the only possible semantics is solution semantics III. Solution semantics II can be used in settings where solutions can be exchanged, but the data exchange constraints and ICs of a peer are not shared with other peers. Finally, the solution semantics I corresponds to a setting in which peers can share the data, the DECs, the ICs and the trust relationships.

The programs for the solutions under semantics I, II and III, with stable model semantics, allow as to obtain the solutions of a peer, and with them, the set of peer consistent answers for UDECs and RDECs. Since we are not interested in the solutions *per se*, but in the PCA, techniques to partially compute them are useful. Techniques used in CQA, such as magic sets, could also be used in this setting to restrict the amount of data considered to run the programs [Caniupan and Bertossi, 2005; Caniupan, 2006]. In [Caniupan and Bertossi, 2005; Caniupan, 2006], many other optimizations can be found that allow for a more efficient evaluation of programs for CQA; they could be explored in the P2P setting.

Obtaining peer consistent answers has at least the data complexity of consistent query answering (CQA), for which some results are known [Chomicki and Marcinkowski, 2002; Chomicki and Marcinkowski, 2005b; Fuxman and Miller, 2003; Calì *et al.*, 2003a; Chomicki and Marcinkowski, 2005a; Fuxman and Miller, 2005]. With CQA, for common database queries and ICs, Π_2^P-completeness is easily achieved. On the other hand, the problem of skeptical query evaluation from the disjunctive programs we are using for P2P data exchange systems is also Π_2^P-complete in data complexity [Dantsin *et al.*, 1997]. In this sense, the logic programs are not contributing with additional complexity to our problem.

With respect to related work, peer-to-peer data exchange systems have been analyzed in [Halevy *et al.*, 2003; Franconi *et al.*, 2004b; Kementsietsidis *et al.*, 2003;

Halevy *et al.*, 2004; Franconi *et al.*, 2004a; Calvanese *et al.*, 2004b; Calvanese *et al.*, 2004a; Calvanese *et al.*, 2005; Fagin *et al.*, 2005] without considering trust relationships. In them, if there is a DEC from P to Q then, implicitly, peer P is assumed to trust itself less than Q. If we restrict ourselves to their setting, the solution semantics I coincides with the solution semantics II. Also, in all the research so far, DECs are considered to force the addition of data to peers. In our setting we also consider that a DEC can restrict the data that can belong to the peer.

Solution semantics I is close in spirit to the semantics in [Halevy *et al.*, 2003] in the sense that both give the semantics putting together the specification of all peers. However, in [Halevy *et al.*, 2003] the specification is done in first-order logic, where each database is a theory and the DECs are first-order formulas.

Solution semantics III can be compared to the semantics in [Franconi *et al.*, 2004b; Franconi *et al.*, 2004a; Calvanese *et al.*, 2004a; Calvanese *et al.*, 2005] in the sense that only the data that the peer is certain of is given to other peers. Their semantics is based in epistemic logic and is able to deal with cyclic DECs.

The results presented in this chapter on local solutions have been published in [Bertossi and Bravo, 2004a], and in a slightly extended version in [Bertossi and Bravo, 2004b]. There, the relationship between P2P systems and data integration systems is also analyzed.

Chapter 8

Conclusions

In this thesis, we have studied consistent query answering from relational databases, data integration systems and peer to peer systems. In all this settings we assume that the databases (either stand-alone, part of a DIS or in a peer) may contain *null* and that we may use it to repair with respect to inconsistencies.

We also analyze the semantics of query answering and satisfaction of integrity constraints in the presence of null values. More specifically, we have proposed a precise and uniform logical reconstruction of IC satisfaction for databases that is compatible with the way null values are treated according to the SQL standard. We also provide a semantics for query answering, called null query answering semantics, that extends the one for IC satisfaction, but that does not always coincide with the query answering semantics of SQL. These results are not only interesting to be used for CQA, but they are interesting by themselves since it is the first logical characterization that is homogenous and extends the portion of the SQL standard implemented in databases. Further research is needed to identify more connections between the null semantics and SQL query answering semantics.

The results obtained for CQA in relational databases assume not only that databases may contain *null* in the form we find them present and treated in current commercial implementations, but that we can also use *null* to restore the consistency of the database. The provided repair semantics applies to a wide class of ICs, including cyclic sets of referential ICs. The CQA can be obtained by using a disjunctive logic

235

program with stable models semantics for RIC-acyclic set of constraints.

In the context of CQA for databases, further research is needed to modify the disjunctive logic program in such a way that it can also provide the CQA for RIC-cyclic set of constraints. It is also of interest, to use disjunctive logic programs with preferences, so that the user can choose different preferences on how the database should be repaired. For example, we might prefer to repair by deletion only certain constraints, or prefer to repair by insertion and only delete if it is not possible to insert.

The connections between CQA in databases with DIS and PDM, allow us to use the results obtained for CQA to these other settings. Table 8.1 shows the parallelism between them.

Database	DIS	PDM
ICs	Mappings and ICs	DECs and ICs
a single database	several database sources	one database per peer
centralized system	centralized mediation system	decentralized mediation system

Table 8.1: Parallel between databases, DISs and PDMs

For data integration systems, we have presented a general approach to specifying, by means of disjunctive logic programs with stable model semantics, the database repairs of a virtual DIS with open sources under the LAV and GAV approaches. Consistent answers to queries posed to such a system are computed by running a query program together with the specification of database repairs and the class of minimal global legal instances of the integration system. To the best of our knowledge, this is also the first specification, under the LAV paradigm, of the global minimal instances in a logic programming formalism. The specification of the minimal legal instances allows us to obtain the *minimal answers* to arbitrary queries; and the *certain answers* to monotone queries, what extends previous results in the literature related to query plan generation under the LAV approach. The specifications given for the GAV

and LAV approaches can be easily combine to provide a specification for the GLAV approach [Friedman *et al.*, 1999].

Research related to the design of virtual data integration systems and its impact on global query answering has been mostly neglected. Most of the research in the area starts from a given set of view definitions, but the conditions on them hardly go beyond classifying them as conjunctive, disjunctive, Datalog, etc. However, other conditions, imposed when the systems is being designed, could have an impact on, e.g. query plan derivation. Much research is needed in this direction.

In the context of P2P data exchange systems, we consider different trust relationships between peers, where each peer solves its conflicts at query time, by querying its own data and neighboring peers' databases. The solution of a peer is not only affected by its neighbors, but also by their interactions with other peers. Since the semantics is given using disjunctive logic programs with stable model semantics, it is possible to retrieve the peer consistent answers for UDECs and RDECs by rewriting the query as a query program.

Further research is needed to determine classes of constraints, DECs, ICs and trust relationships for which the problem of retrieving the PCA can be solved more efficiently. Also research is needed to adjust the optimization techniques used in CQA for stand-alone databases [Caniupan and Bertossi, 2005; Caniupan, 2006] so that they can be used for PDMs.

Some of the results of this thesis have been published in [Barceló *et al.*, 2003; Bravo and Bertossi, 2003; Bravo and Bertossi, 2004; Bertossi and Bravo, 2004a; Bertossi and Bravo, 2004b; Bravo and Bertossi, 2005; Bertossi and Bravo, 2005; Bravo and Bertossi, 2006].

Bibliography

[Abiteboul and Duschka, 1998] Serge Abiteboul and Oliver M. Duschka. Complexity of Answering Queries Using Materialized Views. In *Proceedings of the ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS'98)*, pages 254–263. ACM Press, 1998.

[Abiteboul *et al.*, 1995] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[Arenas *et al.*, 1999] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent Query Answers in Inconsistent Databases. In *Proceedings of the ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS'99)*, pages 68–79. ACM Press, 1999.

[Arenas *et al.*, 2000a] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Specifying and Querying Database Repairs using Logic Programs with Exceptions. In *Proceedings of the International Conference on Flexible Query Answering (FQAS'00)*, pages 27–41. Springer, 2000.

[Arenas *et al.*, 2000b] Marcelo Arenas, Leopoldo Bertossi, and Michael Kifer. Applications of Annotated Predicate Calculus to Querying Inconsistent Databases. In *Proceedings of the International Conference on Computational Logic (CL'00)*, Springer LNAI 1861, pages 926–941, 2000.

[Arenas *et al.*, 2003] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Answer Sets for Consistent Query Answering in Inconsistent Databases. *Theory and Practice of Logic Programming*, 3(4-5):393–424, 2003.

[Atzeni and Morfuni, 1984] Paolo Atzeni and Nicola M. Morfuni. Functional dependencies in relations with null values. *Information Processing Letters*, 18(4):233–238, 1984.

[Atzeni and Morfuni, 1986] Paolo Atzeni and Nicola M. Morfuni. Functional Dependencies and Constraints on Null Values in Database Relations. *Information and Control*, 70(1):1–31, 1986.

[Baral *et al.*, 1992] C. Baral, S. Kraus, J. Minker, and V. S. Subrahmanian. Combining Knowledge Bases Consisting of First-Order Theories. *Computational Intelligence*, 8 (1):45–71, 1992.

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

[Barceló and Bertossi, 2002] Pablo Barceló and Leopoldo Bertossi. Repairing Databases with Annotated Predicate Logic. In *Proceedings of the International Workshop on Non-Monotonic Reasoning (NMR'02), Special session: Changing and Integrating Information*, pages 160–170, 2002.

[Barceló; and Bertossi, 2003] Pablo Barceló; and Leopoldo Bertossi. Logic Programs for Querying Inconsistent Databases. In *Proceedings of the International Symposium on Practical Aspects of Declarative Languages (PADL 03)*, Springer LNCS 2562, pages 208–222, 2003.

[Barceló *et al.*, 2003] Pablo Barceló, Leopoldo Bertossi, and Loreto Bravo. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In Leopoldo Bertossi, Gyula O. H. Katona, Klaus-Dieter Schewe, and Bernhard Thalheim, editors, *Semantics in Databases*, Springer LNCS 2582, pages 7–33, 2003.

[Beeri and Vardi, 1984] Catriel Beeri and Moshe Y. Vardi. A Proof Procedure for Data Dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

[Ben-Eliyahu and Dechter, 1994] Rachel Ben-Eliyahu and Rina Dechter. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.

[Bertossi and Bravo, 2004a] Leopoldo Bertossi and Loreto Bravo. Query Answering in Peer-to-Peer Data Exchange Systems. In *Proceedings EDBT Workshop on Peer-to-Peer Computing and Databases (P2P&DB'04)*, Springer LNCS 3268, pages 476–485, 2004.

[Bertossi and Bravo, 2004b] Leopoldo Bertossi and Loreto Bravo. Query Answering in Peer-to-Peer Data Exchange Systems. Corr archiv, arXiv.org/cs/0401015, 2004.

[Bertossi and Bravo, 2005] Leopoldo Bertossi and Loreto Bravo. Consistent Query Answers in Virtual Data Integration Systems. In Leopoldo Bertossi, Anthony Hunter, and Torsten Schaub, editors, *Inconsistency Tolerance*, Springer LNCS 3300, pages 42–83, 2005.

[Bertossi and Chomicki, 2003] L. Bertossi and J. Chomicki. *Logics for Emerging Applications of Databases*, chapter "Query Answering in Inconsistent Databases", pages 43–83. Springer, 2003.

[Bertossi *et al.*, 2002] Leopoldo Bertossi, Jan Chomicki, Alvaro Cortés, and Claudio Gutiérrez. Consistent Answers from Integrated Data Sources. In *Proceedings of the International Conference Flexible Query Answering Systems (FQAS'02)*, Springer LNCS 2522, pages 71–85, 2002.

[Bertossi *et al.*, 2005a] Leopoldo Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. Complexity and Approximation of Fixing Numerical Attributes in Databases Under Integrity Constraints. In *Proceedings of the International Symposium on Database Programming Languages (DBPL '05)*, Springer LNCS 3774, pages 262–278, 2005.

[Bertossi *et al.*, 2005b] Leopoldo Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. Data Cleansing for Numerical Data Sets. In Andrea Calì, Diego Calvanese, Enrico Franconi, Maurizio Lenzerini, and Letizia Tanca, editors, *Proceedings of the Italian Symposium on Advanced Database Systems (SEBD'05)*, pages 292–299, 2005.

[Bertossi *et al.*, 2005c] Leopoldo Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. Fixing Inconsistent Databases by Updating Numerical Attributes. In *Proceedings of the International Workshop on Logical Aspects and Applications of Integrity Constraints (LAAIC 2005). In Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA'05)*, pages 854–858. IEEE Computer Society, 2005.

[Bertossi *et al.*, 2005d] Leopoldo Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. Fixing Inconsistent Databases by Updating Numerical Attributes. Corr archiv, arXiv.org/cs.DB/0503032, 2005.

[Bertossi, 2006] Leopoldo Bertossi. Consistent Query Answering in Databases. In *ACM Sigmod Record (database principles column)*, volume 2, 2006.

[Bravo and Bertossi, 2003] Loreto Bravo and Leopoldo Bertossi. Logic Programs for Consistently Querying Data Integration Systems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 10–15. Morgan Kaufmann Publishers, 2003.

[Bravo and Bertossi, 2004] Loreto Bravo and Leopoldo Bertossi. Consistent Query Answering under Inclusion Dependencies. In Hanan Lutfiyya, Janice Singer, and Darlene A. Stewart, editors, *CASCON*, pages 202–216. IBM, 2004.

[Bravo and Bertossi, 2005] Loreto Bravo and Leopoldo Bertossi. Deductive Databases for Computing Certain and Consistent Answers from Mediated Data Integration Systems. *Journal of Applied Logic*, 3(1):329–367, 2005.

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

[Buccafurri *et al.*, 2000] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.

[Buneman *et al.*, 1991] Peter Buneman, Achim Jung, and Atsushi Ohori. Using Powerdomains to Generalize Relational Databases. *Theoretical Computer Science*, 91(1):23–55, 1991.

[Calì *et al.*, 2002a] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data Integration under Integrity Constraints. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE'02)*, Springer LNCS 2348, pages 262–279, 2002.

[Calì *et al.*, 2002b] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the Expressive Power of Data Integration Systems. In *Proceedings of the International Conference on Conceptual Modeling (ER'02)*, Springer LNCS 2503, pages 338–350, 2002.

[Calì *et al.*, 2002c] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the Role of Integrity Constraints in Data Integration. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 25(3):39–45, 2002.

[Calì *et al.*, 2003a] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proceedings of the ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS'03)*, pages 260–271. ACM Press, 2003.

[Calì *et al.*, 2003b] Andrea Calì, Domenico Lembo, and Riccardo Rosati. Query Rewriting and Answering under Constraints in Data Integration Systems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 16–21. Morgan Kaufmann Publishers, 2003.

[Calvanese *et al.*, 2003] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query containment. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'03)*, pages 56–67. ACM Press, 2003.

[Calvanese *et al.*, 2004a] Diego Calvanese, Elio Damaggio, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Semantic Data Integration in P2P Systems. In *Proceedings of the VLDB International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'03)*, Springer LNCS 2944, pages 77–90, 2004.

[Calvanese *et al.*, 2004b] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Logical Foundations of Peer-To-Peer Data Integration. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'04)*, pages 241–251. ACM Press, 2004.

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

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

[Caniupan, 2006] Monica Caniupan. *Optimizing And Implementing Repair Programs For Consistent Query Answering In Databases*. PhD thesis, School of Computer Science, Carleton University, 2006.

[Celle and Bertossi, 2000] Alexander Celle and Leopoldo Bertossi. Querying Inconsistent Databases: Algorithms and Implementation. In *Proceedings of the International Conference on Computational Logic (CL'00)*, Springer LNAI 1861, pages 942–956, 2000.

[Chomicki and Marcinkowski, 2002] Jan Chomicki and Jerzy Marcinkowski. Minimal-Change Referential Integrity Maintenance. arXiv.org paper cs.DB/0212004., 2002.

[Chomicki and Marcinkowski, 2005a] Jan Chomicki and Jerzy Marcinkowski. Minimal-Change Integrity Maintenance using Tuple Deletions. *Information and Computation*, 197(1-2):90–121, 2005.

[Chomicki and Marcinkowski, 2005b] Jan Chomicki and Jerzy Marcinkowski. On the Computational Complexity of Minimal-Change Integrity Maintenance in Relational Databases. In Leopoldo Bertossi, Anthony Hunter, and Torsten Schaub, editors, *Inconsistency Tolerance*, Springer LNCS 3300, pages 119–150, 2005.

[Codd, 1979] E. F. Codd. Extending the Database Relational Model to Capture more Meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[Dalal, 1992] Mukesh Dalal. Investigations Into a Theory of Knowledge Base Revision: Preliminary Report. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 475–479. AAAI Press, 1992.

[Dantsin *et al.*, 1997] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. In *Proceedings of the IEEE Conference on Computational Complexity*, pages 82–101, 1997.

[Dantsin *et al.*, 2001] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[Date and Warden, 1990] C. J. Date and Andrew Warden. *Relational Database Writings (1985-1989)*, chapter EXISTS Is Not 'Exists!'. Addison-Wesley Longman Publishing Co., Inc., 1990.

[Doan *et al.*, 2003] AnHai Doan, Pedro Domingos, and Alon Halevy. Learning to Match the Schemas of Data Sources. A Multistrategy Approach. *Machine Learning*, 50(3):279–301, 2003.

[Duschka *et al.*, 2000] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. Recursive Query Plans for Data Integration. *Journal of Logic Programming*, 43(1):49–73, 2000.

[Duschka, 1997] Oliver M. Duschka. *Query Planning and Optimization in Information Integration*. PhD thesis, Stanford University, 1997.

[Eiter *et al.*, 2000] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative Problem-Solving in DLV. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, 2000.

[Eiter *et al.*, 2003] Thomas Eiter, Michael Fink, Gianluigi Greco, and Domenico Lembo. Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In *Proceedings of the International Conference on Logic Programming (ICLP'03)*, Springer LNCS 2916, pages 163–177, 2003.

[Fagin *et al.*, 2003a] R. Fagin, P. Kolaitis, Renée J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *Proceedings of the International Conference on Database Theory (ICDT'03)*, pages 207–224. Springer, 2003.

[Fagin *et al.*, 2003b] Ronald Fagin, Phokion Kolaitis, and Lucian Popa. Data exchange: Getting to the Core. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'03)*, pages 90–101. ACM Press, 2003.

[Fagin *et al.*, 2005] Ronald Fagin, P. Kolaitis, Renée J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[Flesca and Greco, 2001] S. Flesca and S. Greco. Rewriting Queries Using Views. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):980–995, 2001.

[Franconi *et al.*, 2004a] Enrico Franconi, Gabriel Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. A Distributed Algorithm for Robust Data Sharing and Updates in P2P Database Networks. In *Proceedings of the EDBT International Workshop on Peer-to-peer Computing and Databases (P2P&DB'04)*, 2004.

[Franconi *et al.*, 2004b] Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Luciano Serafini. A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems. In *Proceedings of the VLDB International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'03)*, Springer LNCS 2944, pages 64–76, 2004.

[Friedman *et al.*, 1999] Marc Friedman, Alon Levy, and Todd Millstein. Navigational Plans for Data Integration. In *Proceedings of the National Conference on Artificial Intelligence (AAAI 99)*, pages 67–73. AAAI/MIT Press, 1999.

[Fuxman and Miller, 2003] Ariel Fuxman and Renée J. Miller. Towards Inconsistency Management in Data Integration Systems. In *Proceedings of the IJCAI Workshop on Information Integration on the Web (IIWeb'03)*, pages 143–148, 2003.

[Fuxman and Miller, 2005] Ariel Fuxman and Renée Miller. First-Order Query Rewriting for Inconsistent Databases. In *Proceedings of the International Conference on Database Theory (ICDT'05)*, Springer LNCS 3363, pages 337–351, 2005.

[Fuxman *et al.*, 2005a] Ariel Fuxman, Elham Fazli, and Renée J. Miller. ConQuer: Efficient Management of Inconsistent Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'05)*, pages 155–166. ACM Press, 2005.

[Fuxman *et al.*, 2005b] Ariel Fuxman, Diego Fuxman, and Renée J. Miller. Conquer: A system for efficient querying over inconsistent databases. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 1354–1357. ACM, 2005.

[Fuxman *et al.*, 2005c] Ariel Fuxman, Phokion Kolaitis, Renée J. Miller, and Wang-Chiew Tan. Peer Data Exchange. In *Proceedings of the ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS'05)*, pages 160–171. ACM Press, 2005.

[Gelder and Topor, 1987] A. Van Gelder and R. Topor. Safety and Correct Translation of Relational Calculus Formulas. In *Proceedings of the ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS'87)*, pages 313–327. ACM Press, 1987.

[Gelder and Topor, 1991] Allen Van Gelder and Rodney W. Topor. Safety and translation of relational calculus. *ACM Transactions on Database Systems*, 16(2):235–278, 1991.

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

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In *Proceedings of the International Conference on Logic Programming*, pages 1070–1080. The MIT Press, 1988.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[Giannotti *et al.*, 1991] F. Giannotti, D. Pedreschi, D. Saccà, and C. Zaniolo. Non-determinism in Deductive Databases. In *Proceedings of Deductive and Object–Oriented Databases (DOOD '91)*, Springer LNCS 556, pages 129–146, 1991.

[Giannotti *et al.*, 1997] Fosca Giannotti, Sergio Greco, Domenico Sacca, and Carlo Zaniolo. Programming with Non-Determinism in Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 19(1-2):97–125, 1997.

[Grahne and Mendelzon, 1999] Gösta Grahne and Alberto O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. In *Proceeding International Conference on Database Theory (ICDT'99)*, Springer LNCS 1540, pages 332–347, 1999.

[Grahne, 1991] Gösta Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer LNCS 554. 1991.

[Grahne, 2002] Gösta Grahne. Information Integration and Incomplete Information. *Data Engineering Bulletin*, 25(3):46–52, 2002.

[Grant and Minker, 2002] John Grant and Jack Minker. A Logic-Based Approach to Data Integration. *Theory and Practice of Logic Programming*, 2(3):323–368, 2002.

[Grant, 1977] John Grant. Null Values in a Relational Data Base. *Information Processing Letters*, 6(5):156–157, 1977.

[Grant, 1980] John Grant. Incomplete Information in a Relational Database. *Fundamenta Informaticae*, 3(3):363–378, 1980.

[Greco *et al.*, 2001] G. Greco, S. Greco, and E. Zumpano. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *Proceedings of the International Conference on Logic Programming (ICLP'01)*, Springer LNCS 2237, pages 348–364, 2001.

[Gryz, 1999] Jarek Gryz. Query Rewriting Using Views in the Presence of Functional and Inclusion Dependencies. *Information Systems*, 24(7):597–612, 1999.

[Gupta and Mumick, 1995] Ashish Gupta and Inderpal Singh Mumick. Maintenance of Materialized Views: Problems, Techniques and Applications. *IEEE Data Engineering Bulletin*, 18(2):3–18, 1995.

[Gupta and Mumick, 1999] A. Gupta and I.S. Mumick. *Materialized views: techniques, implementations, and applications*. MIT Press Cambridge, MA, USA, 1999.

[Halevy and Madhavan, 2003] Alon Y. Halevy and Jayant Madhavan. Corpus-Based Knowledge Representation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1567–1572. Morgan Kaufmann, 2003.

[Halevy *et al.*, 2003] Alon Halevy, Zachary Ives, Dan Suciu, and Igor Tatarinov. Schema Mediation in Peer Data Management Systems. In *Proceedings of the International Conference on Data Engineering (ICDE'03)*, 2003.

[Halevy *et al.*, 2004] Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. The Piazza Peer Data Management System. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):787–798, 2004.

[Halevy, 2000] Alon Y. Halevy. Theory of Answering Queries Using Views. *SIGMOD Record*, 29(4):40–47, 2000.

[Halevy, 2001] Alon Y. Halevy. Answering Queries Using Views: A Survey. *The International Journal on Very Large Data Bases*, 10(4):270–294, 2001.

[Hull, 1997] Richard Hull. Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. In *Proceedings of the ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS'97)*, pages 51–61. ACM Press, 1997. Invited Tutorial.

[IBM, 2006] IBM. *DB2 UDB SQL Reference, Volume 1, V8*, 2006.

[Imielinski and Lipski, 1984] Tomasz Imielinski and Witold Lipski. Incomplete Information in Relational Databases. *Journal of the ACM*, 31(4):761–791, 1984.

[International Organization for Standardization, 2003] International Organization for Standardization. *ISO International Standard: Database Language SQL - Part 2:SQL/Foundation*, volume 9075. ISO/IEC, 2003.

[Kementsietsidis *et al.*, 2003] Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*, pages 325–336. ACM Press, 2003.

[Kolaitis and Vardi, 2000] P. Kolaitis and M. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences*, 61:302–332, 2000.

[Kolaitis *et al.*, 2006] Phokion Kolaitis, Jonathan Panttaja, and Wang Chiew Tan. The Complexity of Data Exchange. In *Proceedings of the ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS'06)*, pages 30–39, 2006.

[Kripke, 1971] S. A. Kripke. Semantical Considerations on Modal Logic. In L. Linsky, editor, *Reference and Modality*, pages 63–73. Oxford University Press, 1971.

[Lembo *et al.*, 2002] D. Lembo, M. Lenzerini, and R. Rosati. Source Inconsistency and Incompleteness in Data Integration. In *Proceedings of the International Workshop on Knowledge Representation meet Databases (KRDB'02)*, 2002.

[Lenzerini, 2002] Maurizio Lenzerini. Data Integration: a Theoretical Perspective. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems: (PODS'02)*, pages 233–246. ACM Press, 2002.

[Leone *et al.*, 1997] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation*, 135(2):69–112, 1997.

[Leone *et al.*, 2006] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Christoph Koch, Cristinel Mateis, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[Levene and Loizou, 1997a] Mark Levene and George Loizou. Null Inclusion Dependencies in Relational Databases. *Information and Computation*, 136(2):67–108, 1997.

[Levene and Loizou, 1997b] Mark Levene and George Loizou. The Additivity Problem for Functional Dependencies in Incomplete Relations. *Acta Informatica*, 34(2):135–149, 1997.

[Levene and Loizou, 1999a] Mark Levene and George Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer-Verlag, 1999.

[Levene and Loizou, 1999b] Mark Levene and George Loizou. Database Design for Incomplete Relations. *ACM Transactions on Database Systems*, 24(1):80–126, 1999.

[Levy *et al.*, 1995] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering Queries Using Views. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'95)*, pages 95–104, 1995.

[Levy *et al.*, 1996] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the International Conference on Very Large Databases (VLDB'96)*, pages 251–262, 1996.

[Levy, 2000] Alon Y. Levy. Logic-Based Techniques in Data Integration. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 575–595. Kluwer Academic Publishers, 2000.

[Libkin, 1991] Leonid Libkin. A Relational Algebra for Complex Objects based on Partial Information. In *Proceedings of the Symposium on Mathematical Fundamentals of Database Systems (MFDBS'91)*, Springer LNCS 495, pages 136–147, 1991.

[Libkin, 1995] Leonid Libkin. A Semantics-based Approach to Design of Query Languages for Partial Information. In Leonid Libkin and Bernhard Thalheim, editors, *Semantics in Databases*, Springer LNCS 1358, pages 170–208, 1995.

[Lien, 1979] Y. Edmund Lien. Multivalued Dependencies with Null Values in Relational Data Bases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'79)*, pages 61–66. IEEE Computer Society Press, 1979.

[Lien, 1982] Y. Edmund Lien. On the Equivalence of Database Models. *Journal of the ACM*, 29(2):333–362, 1982.

[Lifschitz and Turner, 1994] Vladimir Lifschitz and Hudson Turner. Splitting a Logic Program. In *Proceedings of the International Conference on Logic Programming*, pages 23–37, 1994.

[Lin and Mendelzon, 1998] Jinxin Lin and Alberto O. Mendelzon. Merging Databases Under Constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1998.

[Lloyd, 1987] J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1987.

[Lopatenko and Bertossi, 2006a] Andrei Lopatenko and Leopoldo Bertossi. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. Corr archiv, arXiv.org/cs/0604002, 2006.

[Lopatenko and Bertossi, 2006b] Andrei Lopatenko and Leopoldo Bertossi. Consistent Query Answering By Minimal-Size Repairs. In *Proceedings of the International Workshop on Logical Aspects and Applications of Integrity Constraints (LAAIC 2006). In Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA'06)*, pages 558–562. IEEE Computer Society, 2006.

[Lopatenko and Bravo, 2006] Andrei Lopatenko and Loreto Bravo. Efficient Approximation Algorithms for Repairing Inconsistent Databases. Conference submision, July 2006.

[McBrien and Poulovassilis, 2003] Peter McBrien and Alexandra Poulovassilis. Data Integration by Bi-Directional Schema Transformation Rules. In *Proceedings of the International Conference on Data Engineering (ICDE'03)*, pages 227–238, 2003.

[Microsoft, 2006] Microsoft. *SQL Server 2005*, July 2006.

[Millstein *et al.*, 2003] T. Millstein, A. Halevy, and M. Friedman. Query containment for data integration systems. *Journal of Computer and System Sciences*, 66(1):20–39, 2003.

[MySQL, 2006] MySQL. *Reference Manual versions 3.23, 4.0 and 4.1*, 2006.

[Oracle, 2005] Oracle. *SQL Reference, 10g Release 2 (10.2)*, 2005.

[PostgreSQL, 2006] PostgreSQL. *8.2.0 Documentation*, 2006.

[Pottinger and Bernstein, 2002] Rachel Pottinger and Philip A. Bernstein. Creating a Mediated Schema Based on Initial Correspondences. *IEEE Data Engineering Bulletin*, 25(3):26–31, 2002.

[Przymusinski, 1991] Teodor C. Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9(3/4):401–424, 1991.

[Rahm and Bernstein, 2001] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The International Journal on Very Large Data Bases*, 10(4):334–350, 2001.

[Reiter, 1984] Raymond Reiter. Towards a Logical Reconstruction of Relational Database Theory. In Michael L. Brodie, John Mylopoulos, and Joachim W. Schmidt, editors, *On Conceptual Modelling*, pages 191–233. Springer-Verlag, 1984.

[Reiter, 1986] R. Reiter. A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values. *Journal of the ACM*, 33(2):349–370, 1986.

[Sybase, 2006] Sybase. *Transact-SQL User's Guide, Adaptive Server Enterprise 15.0*, 2006.

[Türker and Gertz, 2001] Can Türker and Michael Gertz. Semantic Integrity Support in SQL:1999 and Commercial (Object-)Relational Database Management Systems. *The International Journal on Very Large Data Bases*, 10(4):241–269, 2001.

[Ullman and Widom, 2002] J.D. Ullman and J. Widom. *A First Course in Database Systems*. Prentice Hall, second edition, 2002.

[Ullman, 1988] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.

[Ullman, 2000] Jeffrey D. Ullman. Information Integration using Logical Views. *Theoretical Computer Science*, 239(2):189–210, 2000.

[van der Meyden, 1998] Ron van der Meyden. Logical Approaches to Incomplete Information: A Survey. In Jan Chomicki and Gunter Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.

[Wang and Zaniolo, 2000] Haixun Wang and Carlo Zaniolo. Nonmonotonic Reasoning in $\mathcal{LDL}^{++}$. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 523–544. Kluwer Academic Publishers, 2000.

[Wijsen, 2003] Jef Wijsen. Condensed Representation of Database Repairs for Consistent Query Answering. In *Proceedings of the International Conference on Database Theory (ICDT'03)*, Springer LNCS 2572, pages 378–393, 2003.

[Wijsen, 2005] Jef Wijsen. Database Repairing using Updates. *ACM Transactions on Database Systems*, 30(3):722–768, 2005.

[Yang and Garcia-Molina, 2003] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proceedings of the International Conference on Data Engineering (ICDE'03)*, page 49. IEEE Computer Society, 2003.

[Zaniolo, 1984] C. Zaniolo. Database Relations with Null Values. *Journal of Computer and System Sciences*, 28(1):142–166, 1984.

Appendix A

Optimizations of $\Pi(D, IC)$

The logic programs used to specify database repairs can be optimized in several ways. First, we should try to avoid the explicit computation of negative information, done by rule $P(\bar{x}, \mathbf{f}^\star) \leftarrow dom(\bar{x}),\ not\ P(\bar{x}, \mathbf{t_d})$. We would like to calculate only the negative information that is explicitly needed to find inconsistencies and solve them. Second, we would like to get rid of unneeded predicates, such as $dom(x)$. Finally we will combine different rules to minimize also the number of rules.

In order to avoid the computation of negative information that is not needed, we can modify the program by:

1. First, by unfolding, atoms of the form $P(\bar{x}, \mathbf{f}^\star)$ that appear as subgoals in the bodies are replaced by their definitions. More precisely, replace every rule that contains an atom of the form $P(\bar{x}, \mathbf{f}^\star)$ in the body, by two rules, one replacing the atom by $P(\bar{x}, \mathbf{f_a})$, and another replacing the atom by $not\ P(\bar{x}, \mathbf{t_d})$.

2. Next, eliminate from the repair program those rules that have atoms annotated with $\mathbf{f}^{\star\star}$ or $\mathbf{f}^\star$ in their heads, because they compute data that should not be explicitly contained in the repairs.

Let $\Pi^\star(D, IC)$ denote the program obtained after applying these two transformations.

Example A.1 Consider the database instance $\{P(a, b)\}$ that is inconsistent with respect to the constraint $\forall x, y(P(x, y) \to P(y, x))$. The program $\Pi(D, IC)$ corresponds to:

$dom(a). \qquad dom(b). \qquad \underline{P}(a, b, \mathbf{t_d}).$

$\underline{P}(x, y, \mathbf{f_a}) \vee \underline{P}(y, x, \mathbf{t_a}) \leftarrow \underline{P}(x, y, \mathbf{t}^\star), \underline{P}(y, x, \mathbf{f}^\star), dom(x), dom(y).$

$\underline{P}(x, y, \mathbf{t}^\star) \leftarrow \underline{P}(x, y, \mathbf{t_a}).$

$\underline{P}(x, y, \mathbf{t}^\star) \leftarrow \underline{P}(x, y, \mathbf{t_d}).$

$\underline{P}(x, y, \mathbf{f}^\star) \leftarrow \underline{P}(x, y, \mathbf{f_a}).$

$\underline{P}(x, y, \mathbf{f}^\star) \leftarrow dom(x), dom(y), \ not \ \underline{P}(x, y, \mathbf{t_d}).$

$\underline{P}(x, y, \mathbf{t}^{\star\star}) \leftarrow \underline{P}(x, y, \mathbf{t_a}).$

$\underline{P}(x, y, \mathbf{t}^{\star\star}) \leftarrow \underline{P}(x, y, \mathbf{t_d}), \ not \ \underline{P}(x, y, \mathbf{f_a}).$

$\underline{P}(x, y, \mathbf{f}^{\star\star}) \leftarrow \underline{P}(x, y, \mathbf{f_a}).$

$\underline{P}(x, y, \mathbf{f}^{\star\star}) \leftarrow dom(x), dom(y), \ not \ \underline{P}(x, y, \mathbf{t_d}), \ not \ \underline{P}(x, y, \mathbf{t_a}).$

$\leftarrow \underline{P}(x, y, \mathbf{t_a}), \underline{P}(x, y, \mathbf{f_a}).$

The optimized repair program $\Pi^\star(D, IC)$ is:

$dom(a). \qquad dom(b). \qquad \underline{P}(a, b, \mathbf{t_d}).$

$\underline{P}(x, y, \mathbf{f_a}) \vee \underline{P}(y, x, \mathbf{t_a}) \leftarrow \underline{P}(x, y, \mathbf{t}^\star), \ not \ \underline{P}(y, x, \mathbf{t_d}), dom(x), dom(y).$

$\underline{P}(x, y, \mathbf{f_a}) \vee \underline{P}(y, x, \mathbf{t_a}) \leftarrow \underline{P}(x, y, \mathbf{t}^\star), \underline{P}(y, x, \mathbf{f_a}), dom(x), dom(y).$

$\underline{P}(x, y, \mathbf{t}^\star) \leftarrow \underline{P}(x, y, \mathbf{t_a}).$

$\underline{P}(x, y, \mathbf{t}^\star) \leftarrow \underline{P}(x, y, \mathbf{t_d}).$

$\underline{P}(x, y, \mathbf{t}^{\star\star}) \leftarrow \underline{P}(x, y, \mathbf{t_a}).$

$\underline{P}(x, y, \mathbf{t}^{\star\star}) \leftarrow \underline{P}(x, y, \mathbf{t_d}), \ not \ \underline{P}(x, y, \mathbf{f_a}).$

$\leftarrow \underline{P}(x, y, \mathbf{t_a}), \underline{P}(x, y, \mathbf{f_a}).$

Note that now we only use 5 annotation constants and no negative information is materialized (there are no $\mathbf{f}^\star$ and $\mathbf{f}^{\star\star}$). □

Example A.2 (example 5.10 continued) The optimized program $\Pi^\star(D, IC)$ is as below and determines the same repairs as the original program. Notice that the second disjunctive rule in the original program was replaced by two new rules in the new program.

$dom(a).$

$P\_(a, \mathbf{t_d}).$

$P\_(x, \mathbf{f_a}) \vee Q\_(x, null, \mathbf{t_a}) \leftarrow P\_(x, \mathbf{t^\star}), \ not \ aux(x), dom(x).$

$aux(x) \leftarrow Q\_(x, y, \mathbf{t^\star}), \ not \ Q\_(x, y, \mathbf{f_a}), dom(x), dom(y).$

$aux(x) \leftarrow Q(x, null), \ not \ Q\_(x, null, \mathbf{f_a}), dom(x).$

$Q\_(x, y, \mathbf{f_a}) \vee R\_(x, y, \mathbf{t_a}) \leftarrow Q\_(x, y, \mathbf{t^\star}), R\_(x, y, \mathbf{f_a}), dom(x), dom(y).$

$Q\_(x, y, \mathbf{f_a}) \vee R\_(x, y, \mathbf{t_a}) \leftarrow Q\_(x, y, \mathbf{t^\star}), \ not \ R\_(x, y, \mathbf{t_d}), dom(x), dom(y).$

$$\left.\begin{array}{l} P\_(x, \mathbf{t^\star}) \leftarrow P\_(x, \mathbf{t_a}). \\[4pt] P\_(x, \mathbf{t^\star}) \leftarrow P\_(x, \mathbf{t_d}). \\[4pt] P\_(x, \mathbf{t^{\star\star}}) \leftarrow P\_(x, \mathbf{t_a}). \\[4pt] P\_(x, \mathbf{t^{\star\star}}) \leftarrow P\_(x, \mathbf{t_d}), \ not \ P\_(x, \mathbf{f_a}). \\[4pt] \leftarrow P\_(x, \mathbf{t_a}), P\_(x, \mathbf{f_a}). \end{array}\right\} \quad \text{(Similarly for } Q \text{ and } R)$$

This optimized repair program calculates negative information only when needed □

After modifying the program not to materialize the negative, the only objective of predicate $dom(x)$ is to enforce that variable x cannot be *null*. Therefore, we could avoid the use of this predicate by replacing every occurrence of $dom(x)$ by $x \neq null$.

On the other hand, we would like to get rid of the annotation $\mathbf{t_d}$. This would be useful since it will allow to separate the intentional (the rules) from the extensional database (the data itself). This can be simply done by replacing every predicate $P\_(\bar{x}, \mathbf{t_d})$ by $P(\bar{x})$. This simple change allows, for example, to include in DLV only the rules and leave the facts in the database. The database will be accessed by DLV to retrieve only the data that is needed.

Example A.3 (example 5.11 continued) The following corresponds to the optimized repair program. For the sake of comparison, the same items as in Example 5.11 were kept.

1. There are no *dom* atoms.

2. $Reg(21, C15).$ $Reg(34, C18).$ $Student(21, Ann).$ $Student(45, Paul).$

3. There is no UIC.

4. $Reg\_(x, y, \mathbf{f_a}) \vee Student\_(x, null, \mathbf{t_a}) \leftarrow Reg\_(x, y, \mathbf{t^\star}), not\ aux(x),\ x \neq null.$

   $aux(x) \leftarrow Student\_(x, y, \mathbf{t^\star}),\ not\ Student\_(x, y, \mathbf{f_a}),\ x \neq null,\ y \neq null.$

   $aux(x) \leftarrow Student(x, null),\ not\ Student\_(x, null, \mathbf{f_a}),\ x \neq null.$

5. $Reg\_(x, y, \mathbf{t^\star}) \leftarrow Reg\_(x, y, \mathbf{t_a}).$

   $Reg\_(x, y, \mathbf{t^\star}) \leftarrow Reg(x, y).$

   $Student\_(x, y, \mathbf{t^\star}) \leftarrow Student\_(x, y, \mathbf{t_a}).$

   $Student\_(x, y, \mathbf{t^\star}) \leftarrow Student(x, y).$

6. $Reg\_(x, y, \mathbf{t^{\star\star}}) \leftarrow Reg\_(x, y, \mathbf{t_a}).$

   $Reg\_(x, y, \mathbf{t^{\star\star}}) \leftarrow Reg(x, y),\ not\ Reg\_(x, y, \mathbf{f_a}).$

   $Student\_(x, y, \mathbf{t^{\star\star}}) \leftarrow Student\_(x, y, \mathbf{t_a}).$

   $Student\_(x, y, \mathbf{t^{\star\star}}) \leftarrow Student(x, y),\ not\ Student\_(x, y, \mathbf{f_a}).$

7. $\leftarrow Reg\_(x, y, \mathbf{t_a}), Reg\_(x, y, \mathbf{f_a}).$

   $\leftarrow Student\_(x, y, \mathbf{t_a}), Student\_(x, y, \mathbf{f_a}).$ $\qquad\qquad\qquad\qquad$ □

$\Pi^\star(D, IC)$ is the optimized program $\Pi(D, IC)$ modified so that the negative data is not materialized, without predicate $dom(x)$ and without annotation constant $\mathbf{t_d}$. $\Pi^\star(D, IC)$ is described in Definition 5.9.

Example A.4 Consider $D = \{P(a, b), P(c, null)\}$ and the UIC: $\forall xy(P(x, y) \rightarrow R(x) \vee S(y))$. Then $\Pi^\star(D, IC)$:

1. $P(a, b).$ $P(c, null).$

2. $P\_(x, y, \mathbf{f_a}) \vee R\_(x, \mathbf{t_a}) \vee S\_(y, \mathbf{t_a}) \leftarrow P\_(x, y, \mathbf{t^\star}), R\_(x, \mathbf{f_a}), S\_(y, \mathbf{f_a}), x \neq null, y \neq null.$

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

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

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

3. There is no RIC

4. $P\_(x, y, \mathbf{t^\star}) \leftarrow P\_(x, y, \mathbf{t_a}).$

$P\_(x, y, \mathbf{t^\star}) \leftarrow P\_(x, y, \mathbf{t_d}).$

$P\_(x, y, \mathbf{t^{\star\star}}) \leftarrow P\_(x, y, \mathbf{t_a}).$ (Similarly for R and S)

$P\_(x, y, \mathbf{t^{\star\star}}) \leftarrow P\_(x, y, \mathbf{t_d}), \ not\ P\_(x, y, \mathbf{f_a}).$

$\leftarrow P\_(x, y, \mathbf{t_a}), P\_(x, y, \mathbf{f_a}).$

The rules in 2. are constructed by choosing all the possible sets Q' and Q'' such that $Q' \cup Q'' = \{R(x), S(y)\}$ and $Q' \cap Q'' = \emptyset$. The first rule in 2. corresponds to $Q' = \{R(x), S(y)\}$ and $Q'' = \emptyset$, the second for $Q' = \{R(x)\}$ and $Q'' = \{S(y)\}$, the third for $Q' = \{S(y)\}$ and $Q'' = \{R(x)\}$, and the fourth for $Q' = \emptyset$ and $Q'' = \{R(x), S(y)\}$ □

Proposition A.1 $\Pi^\star(D, IC)$ and $\Pi(D, IC)$ produce the same database repairs, more precisely, they compute exactly the same database instances in the sense of Definition 5.7. □

Other possible optimizations, that are not further discussed here, have to do with avoiding the complete computation of all stable models (the repairs) whenever a query is to be answered. The query rewriting methodology introduced in [Arenas *et al.*, 1999] had this advantage: inconsistencies were solved locally, without having to restore the consistency of the complete database. In contrast, the logic programming base methodology, at least if implemented in a straightforward manner, computes each

stable model completely. This issue is related to finding methodologies for minimizing the number of rules to be instantiated, avoiding evaluation of irrelevant subgoals, etc. These type of optimizations are addressed in [Caniupan and Bertossi, 2005; Caniupan, 2006]. Also, results from evaluation of logic programs for data integration systems [Eiter *et al.*, 2003] can be applied in the context of single relations databases.

Appendix B

Simple Program obtained from the Refined Program

Under the hypothesis of Theorem 6.2, there is a simple syntactic transformation of the refined program into a simple program (in the sense of Section 6.2.1) that has the same stable models, and then, in particular, produces the same database instances.

Assume the hypothesis of Theorem 6.2 hold. We denote the view sections with S_i^l as in Section 6.2.1. The sections S_i^l are all associated to the definition of view V_i. We show now a syntactic transformation of the refined version of the program $\Pi(\mathcal{G})$. We justify each step of the transformation, so that at the end it will be clear that they have the same models.

Since there is no admissible mapping, each S_i^l can only be generated by view V_i. As a consequence, for every model $\mathcal{M}$ of the refined version of $Pi(\mathcal{G})$, it holds that for all $\bar{a}$, $var_{v_{ij}z_l}(\bar{a}) \notin \mathcal{M}$. This implies that for every model $\mathcal{M}$ and $\bar{a}$, $aux_{v_{ij}}(\bar{a}) \notin \mathcal{M}$ and $aux_{v_{ij}z_l}(\bar{a}) \notin \mathcal{M}$. Since those atoms will never appear in a model of the refined version of $Pi(\mathcal{G})$, we can delete the rules with those predicates in their heads. We can also delete them from the bodies of the rules where they appear negated. We obtain the following program:

1. Fact $dom(a)$ for every constant $a \in \mathcal{U}$.

2. Fact $V_i(\bar{a})$ whenever $\bar{a} \in v_i$ for some source extension v_i in $\mathcal{G}$.

3. For every view (source) predicate V_i in the system with description $V_i(\bar{x}) \leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$:

(a) For every P_k with no existential variables, the rules

$$P_{k\_}(\bar{x}_k, \mathbf{t_o}) \leftarrow V_i(\bar{x}).$$

(b) For every set S_{ij} of predicates of the description's body that are related by common existential variables $\{z_1, \ldots, z_m\}$, the rules,

$$P_{k\_}(\bar{x}_k, \mathbf{v_{ij}}) \leftarrow add_{v_{ij}}(\bar{x}'), \ \bigwedge_{z_l \in (\bar{x}_k \setminus \bar{x}')} F_i^l(\bar{x}', z_l), \text{ for } P_k \in S_{ij}.$$

$$add_{v_{ij}}(\bar{x}') \leftarrow V_i(\bar{x}), \text{ where } \bar{x}' = \bar{x} \cap \{\textstyle\bigcup_{P_k \in S_{ij}} \bar{x}_k\}.$$

4. For every predicate $F_i^l(\bar{x}', z_l)$ introduced in 3.b., the rules,

$$F_i^l(\bar{x}', z_l) \leftarrow add_{v_{ij} z_l}(\bar{x}'), dom(z_l), choice((\bar{x}'), (z_l)).$$

$$add_{v_{ij} z_l}(\bar{x}') \leftarrow add_{v_{ij}}(\bar{x}'), \text{ for } l = 1, \ldots m.$$

5. For every global relation $P(\bar{x})$ the rules

$$P_{\_}(\bar{x}, \mathbf{nv_{ij}}) \leftarrow P_{\_}(\bar{x}, \mathbf{v_{hk}}), \quad \text{for } \{(ij, hk) | P(\bar{x}) \in S_{ij} \text{ and } S_{hk}\}.$$

$$P_{\_}(\bar{x}, \mathbf{nv_{ij}}) \leftarrow P_{\_}(\bar{x}, \mathbf{t_o}), \quad \text{for } \{(ij) | P(\bar{x}) \in S_{ij}\}.$$

$$P(\bar{x}) \leftarrow P_{\_}(\bar{x}, \mathbf{v_{ij}}), \quad \text{for } \{(ij) | P(\bar{x}) \in S_{ij}\}.$$

$$P(\bar{x}) \leftarrow P_{\_}(\bar{x}, \mathbf{t_o}).$$

This is a positive program with choice. Because of the second rule in 3.(b) and the second rule in 4., we can replace every occurrence of $add_{v_{ij}}(\bar{x}')$ and $add_{v_{ij} z_l}(\bar{x}')$ by $V_i(\bar{x})$. Also from the third and fourth rules in 5., we can replace every occurrence of $P_{\_}(\bar{x}, \mathbf{t_o})$ and $P_{\_}(\bar{x}, \mathbf{v_{ij}})$ by $P(\bar{x})$. It is also easy to see that the first two rules in 5. will generate atoms that are useless in the calculation of the global predicates; then these rules can be deleted. We obtain the following program:

1. Fact $dom(a)$ for every constant $a \in \mathcal{U}$.

2. Fact $V_i(\bar{a})$ whenever $\bar{a} \in v_i$ for some source extension v_i in $\mathcal{G}$.

3. For every view (source) predicate V_i in the system with description $V_i(\bar{x}) \leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$:

   (a) For every P_k with no existential variables, the rules

   $$P_k(\bar{x}_k) \leftarrow V_i(\bar{x}).$$

   (b) For every set S_{ij} of predicates of the description's body that are related by common existential variables $\{z_1, \ldots, z_m\}$, the rules,

   $$P_k(\bar{x}_k) \leftarrow V_i(\bar{x}), \ \bigwedge_{z_l \in (\bar{x}_k \setminus \bar{x}')} F_i^l(\bar{x}', z_l), \text{ for } P_k \in S_{ij}.$$

4. For every predicate $F_i^l(\bar{x}', z_l)$ introduced in 3.b., the rules,

$$F_i^l(\bar{x}', z_l) \leftarrow V_i(\bar{x}), dom(z_l), choice((\bar{x}'), (z_l)).$$

By merging rules 3.(a) and 3.(b), the revised version of $\Pi(\mathcal{G})$ is eventually syntactically transformed into the simple version of the program.