Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Applied Logic 3 (2005) 329–367

JOURNAL OF APPLIED LOGIC

www.elsevier.com/locate/jal

Deductive databases for computing certain and consistent answers from mediated data integration systems

Loreto Bravo [*], Leopoldo Bertossi

Carleton University, School of Computer Science, Ottawa, Canada

Available online 18 August 2004

Abstract

We address the problem of retrieving certain and consistent answers to queries posed to a mediated data integration system under the local-as-view paradigm with open sources and conjunctive and disjunctive view definitions. For obtaining certain answers a query program is run under the cautious stable model semantics on top of a normal deductive database with *choice* operator that specifies the class of minimal legal instances of the integration system. This methodology works for all monotone Datalog queries. To compute answers to queries that are consistent with respect to given global integrity constraints, the specification of minimal legal instances is combined with another disjunctive deductive database that specifies the repairs of those legal instances. This allows to retrieve the answers to any Datalog$^{\neg}$ query that are consistent with respect to global universal and referential integrity constraints.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Databases; Virtual data integration; Integrity constraints; Answer Set Programming

1. Introduction

Usually independent and autonomous data sources are virtually integrated by means of a mediator, which is a program that provides a global schema as an interface, and is

[*] Corresponding author.
E-mail addresses: lbravo@scs.carleton.ca (L. Bravo), bertossi@scs.carleton.ca (L. Bertossi).

1570-8683/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.jal.2004.07.023

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 side, the "Global-As-View" (GAV) approach, defines every global relation as a view of the set of relations in the sources (see [33] for a survey on these and mixed approaches). Query answering is harder under LAV [2]. On the other side, LAV offers more flexibility to accept or release sources into/from an existing system.

In these 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 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) $R: X \rightarrow Y$ is violated through the pair of tuples $\{(a,b), (a,c)\}$.

Inconsistencies are not exclusive to integration systems. For several reasons also single databases may become inconsistent with respect to certain ICs. Restoring consistency may be undesirable, difficult or impossible [10]. In such a situation, possibly most of the data is still consistent and can be retrieved when queries are posed to the database. In [3] consistent data in a stand-alone relational database is characterized as the data that is invariant under all minimal restorations of consistency, i.e., as data that is present in all repaired versions of the original instance (the *repairs*). In particular, an answer to a query is defined as consistent when it can be obtained as a standard answer to the query from every possible repair.

In [3–5,17,30], some mechanisms have been developed for consistent query answering (CQA), i.e., for retrieving consistent answer when queries are posed to such an inconsistent database. All those mechanisms, in different degrees, work only with the original, inconsistent database, without restoring its consistency. That is, inconsistencies are solved at query time. The above mentioned repairs provide an auxiliary concept that allows defining the right semantics for consistent query answers. Furthermore, in some of the query evaluation methodologies, repairs are also an auxiliary computational intermediate step that, for complexity reasons, has to be kept to a minimum.

In virtual data integration systems, there is also an intuitive notion of consistent answer to a query.

Example 2 (*Example 1 continued*). If we pose to the global system the query $Q: Ans(X,Y) \leftarrow R(X,Y)$, 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 $R: X \rightarrow Y$.

Several algorithms for deriving query plans to obtain query answers from virtual data integration systems have been proposed in the last few years (see [36] for a survey). However they are not designed for obtaining the consistent answers to queries. Even more, some of those algorithms assume that certain ICs hold at the global level [21,29,31]; what may not be a realistic assumption due to the independence of the different data sources and the lack of a central, global maintenance mechanism. Only a few exceptions, including this paper, consider the problem of CQA in virtual integration systems [9,13,16,32].

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 [9] a methodology for generating query plans to compute answers to limited forms of queries that are consistent with respect to an also restricted class of universal ICs was presented. This method uses the query rewriting approach to CQA presented in [3]; and in consequence inherits its limitations in terms of the queries and ICs that it can handle, actually queries that are conjunctions of tables and universal ICs. Once the query is transformed, query plans are generated for the new query. However, [9] provides the right semantics for CQA in mediated integrated systems (see Section 2).

In this paper, under the LAV approach and assuming that sources are open (or incomplete) [2], we solve the problem of retrieving consistent answers to global queries. We consider arbitrary universal ICs and referential ICs; that is, the ICs that are most used in database praxis [1]. 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 deductive database with *choice operator* [25] and stable model semantics [24], the class of all minimal legal global instances of a virtual integration system. This approach is inspired by the inverse-rules algorithm [21] and uses auxiliary Skolem predicates whose functionality is enforced with the choice operator.

In order to obtain answers to global queries from the integration system, a query program has to be combined with the deductive database that specifies the minimal instances as its stable models, 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$^{\neg}$ 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 [9]. Consistent answers from an inconsistent integration systems are those that can be obtained from all the repairs of all the minimal legal instances with respect to the global ICs [3,9]. In consequence, 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 given ICs. The latter is a disjunctive deductive database that specifies the repairs as its stable models; and uses annotation constants as in the case of repairs of single relational databases [3] as presented in [5,6]. We have experimented with this query answering mechanism (and the computation of minimal instances and their repairs) with the DLV system [22,35], which implements the stable model and answer set semantics of disjunctive extended deductive databases.

The paper is structured as follows. In Section 2 we review some basic notions we need in the rest of this paper. In Section 3, the minimal legal global instances of a mediated system are specified by means of logic programs with a stable model, or answer sets, semantics. In Section 4, the repairs of the minimal global instances are specified as the stable models of disjunctive logic programs with annotation constants, like those used to specify repairs of single relational databases for CQA [6]. In Section 5, consistent answers to queries are obtained by running a query program in combination with the previous two specification programs. In Section 6 several issues and possible extensions around the specification presented in the previous sections are discussed in detail. Finally, in Section 7, we draw some final conclusions, and we point to related and future work. Appendix A.1 contains the proofs of the main results in this paper.

This paper is an extended version of [13] that now includes the most general specification of minimal instances, the proofs, an extension to disjunctive view definitions, and an analysis of: complexity, the underlying assumptions about the domain, a comparison between the use of the choice operator and the use of Skolem functions.

2. Preliminaries

2.1. Global schemas and view definitions

A *global schema* $\mathcal{R}$ consists of a finite set of relations $\{R_1, R_2, \ldots, R_m\}$ over a fixed, possibly infinite domain $\mathcal{U}$. 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 $V_1, V_2, \ldots, V_n$, 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 [1], 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, and $body(\varphi_V)$ is a conjunction of $\mathcal{R}$-atoms. In general, $V \in \mathcal{S}$.

A *database instance* D over schema $\mathcal{R}$ can be considered as a first-order structure with domain $\mathcal{U}$, where the extensions of the relations R_i are finite. The extensions of built-in predicates may be infinite, but fixed. A global *integrity constraint* (IC) is an $\mathcal{L}(\mathcal{R})$-sentence ψ. An instance D satisfies ψ, denoted $D \models \psi$, if ψ is true in D.

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 υ (corresponding to the contents of a data source), it is possible that υ is incomplete and stores only some of the tuples in $\varphi_V(D)$; i.e., $\upsilon \subseteq \varphi_V(D)$, and we say the view extension υ is *open* with respect to D [2]. Most mechanisms for deriving query plans assume that sources are open, e.g., [21].

A *source* S is a pair $\langle \varphi, \upsilon \rangle$, where φ is the view definition, and υ 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 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 [33].

Definition 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 $\mathit{Linst}(\mathcal{G}) = \{D \text{ instance over } \mathcal{R} \mid v_i \subseteq \varphi_i(D),\ i = 1, \ldots, n\}$.

Example 3 (*Example 2 continued*). Let us denote by φ_1, φ_2 the view definitions of V_1, V_2, resp. in $\mathcal{G}_1$. $D = \{R(a,b), R(c,d), R(a,c), R(d,e)\}$ 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.

The semantics of query answers in mediated integration systems is given by the notion of *certain answer*. In this paper we will consider queries expressed in Datalog and its extensions with negation.

Definition 2 [2]. 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 $D \in \mathit{Linst}(\mathcal{G})$, it holds $D \models Q[\bar{t}]$.[1] We denote with $\mathit{Certain}_{\mathcal{G}}(Q)$ the set of certain answers to Q in $\mathcal{G}$.

The inverse-rules algorithm [21] 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 to 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 directly applied. 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 4. Consider the integration system $\mathcal{G}_2$ with global schema $\mathcal{R} = \{P, R\}$. The set $\mathcal{V}$ of local view definitions consists of $V_1(X,Z) \leftarrow P(X,Y), R(Y,Z)$, and $V_2(X,Y) \leftarrow P(X,Y)$. 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 $V_1(X,Z) \leftarrow$

[1] $D \models Q[\bar{t}]$ means that query $Q(\bar{X})$ becomes true in instance D, when tuple of variables $\bar{X}$ is assigned the values in the tuple $\bar{t}$ of database elements.

$P(X,Y), R(Y,Z,W)$, 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 $R(f(X,Z), Z, g(X,Z)) \leftarrow V_1(X,Z)$.

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 [21], and the answers it produces coincide with the certain answers [2].

2.2. Global systems and consistency

We assume that we have a set of global integrity constraints $IC \subseteq \mathcal{L}(\mathcal{R})$ that is consistent as a set of logical sentences, and *generic*, in the sense that it does not entail any ground database literal by itself, i.e., independently of concrete instance [10]. ICs used in database praxis are always generic. The ICs can be universal, i.e., a sentence of the form $\bar{\forall}\varphi$, where $\bar{\forall}$ is a prefix of universal quantifiers and φ a quantifier-free formula; or referential, i.e., of the form

$$\forall \bar{X}\big(P(\bar{X}) \rightarrow \exists Y\, Q(\bar{X}', Y)\big), \quad \bar{X}' \subseteq \bar{X}.^{2} \tag{1}$$

Definition 3 [9]. (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 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 5 (*Example 4 continued*). Assume that $\mathcal{G}_2$ has the source contents $v_1 = \{V_1(a,b)\}$, $v_2 = \{V_2(a,c)\}$, and that $\mathcal{U} = \{a,b,c,u,\ldots\}$. 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 $P(X,Y): X \rightarrow Y$ is violated exactly in those minimal legal instances D_z for which $z \neq c$. Thus, $\mathcal{G}_2$ is inconsistent.

Definition 4 [9]. The ground tuple $\bar{a}$ is a *minimal answer* to a query Q posed to $\mathcal{G}$ if for every $D \in Mininst(\mathcal{G})$, $\bar{a} \in Q(D)$, where $Q(D)$ is the answer set for Q in D. The set of minimal answers is denoted by $Minimal_{\mathcal{G}}(Q)$.

Clearly $Certain_{\mathcal{G}}(Q) \subseteq Minimal_{\mathcal{G}}(Q)$. For monotone queries [1], the two notions coincide [9]. Nevertheless, in Example 5 the query $Ans(X,Y) \leftarrow \neg P(X,Y)$ 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.

[2] To keep the presentation simple, Y is a single variable, however it could be a tuple of variables, actually interleaved with those in $\bar{X}'$.

Definition 5 [3]. (a) Given a database instance D, we denote by $\Sigma(D)$ the set of ground atomic formulas $\{P(\bar{a}) \mid P \in \mathcal{R} \text{ and } D \models P(\bar{a})\}$.

(b) 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 $\Delta(D, D') = (\Sigma(D) \setminus \Sigma(D')) \cup (\Sigma(D') \setminus \Sigma(D))$.

We may assume that the original data sources and the global legal instances do not contain null values, however when dealing with referential integrity constraints (RICs), we will consider the possibility of having them, in order to restore the consistency of the database. If no RICs are present, we will assume that null values are not available either. However, if necessary, the null value *null* will be treated as a new, special constant. Its presence in a tuple means that there is an unknown value for the correspondent attribute, i.e., we have incomplete information. Since we do not have precise information about it, we will consider that no inconsistencies arise due to its presence. This leads to the following definition of consistency in the presence of null values:

Definition 6 [6]. For a database instance D, whose domain $\mathcal{U}$ may contain the constant *null* and a set of integrity constraints $IC = IC_U \cup IC_R$, where IC_U is a set of universal integrity constraints and IC_R is a set of referential integrity constraints, we say that D satisfies IC, written $D \models IC$, iff:

(1) For each $\bar{\forall}\varphi \in IC_U$, $D \models \varphi[\bar{a}]$ for every ground tuple $\bar{a}$ of elements in $(\mathcal{U} - \{\mathit{null}\})$, and
(2) For each sentence in IC_R of the form (1), if $D \models P[\bar{a}]$, with $\bar{a}$ a ground tuple of elements in $(\mathcal{U} - \{\mathit{null}\})$, then $D \models \exists Y\, Q(\bar{a}, Y)$.

Example 6. 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 $D = \{P(a, d), R(a, d), T(a), T(b), P(b, \mathit{null})\}$ is consistent. The universal constraint is satisfied even in the presence of $P(b, \mathit{null})$ since the incomplete information cannot generate inconsistencies.

Definition 7 [6]. Let D, D', D'' be database instances over the same schema and domain $\mathcal{U}$. It holds $D' \leqslant_D D''$ iff:

(1) For every atom $P(\bar{a}) \in \Delta(D, D')$, with $\bar{a} \in (\mathcal{U} - \{\mathit{null}\})$,[3] it holds $P(\bar{a}) \in \Delta(D, D'')$, and
(2) For every atom $Q(\bar{a}, \mathit{null}) \in \Delta(D, D')$, it holds $Q(\bar{a}, \mathit{null}) \in \Delta(D, D'')$ or $Q(\bar{a}, b) \in \Delta(D, D'')$ with $b \in (\mathcal{U} - \{\mathit{null}\})$.

Definition 7 defines which databases are closer to the original one in the presence of *null* values. This partial order is used in the next definition for repairs in the presence of universal and referential ICs.

[3] That $\bar{a} \in (\mathcal{U} - \{\mathit{null}\})$ means that each of the elements in tuple $\bar{a}$ belongs to $(\mathcal{U} - \{\mathit{null}\})$.

Definition 8 (*Based on* [3]). 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 $D' \models IC$ and D' is $\leqslant_D$-minimal for some $D \in \mathit{Mininst}(\mathcal{G})$.

According to this definition the repairs of violations of referential ICs are obtained by either deleting the atom that is generating the inconsistency or by adding an atom with a *null* value. In particular, if the instance D is $\{P(\bar{a})\}$ and IC contains only $\forall \bar{x}(P(\bar{x}) \rightarrow \exists y Q(\bar{x}, y))$, then $\{P(\bar{a}), Q(\bar{a}, \mathit{null})\}$ will be a repair, but not $\{P(\bar{a}), Q(\bar{a}, b)\}$, with $b \in \mathcal{U}$ and $b \neq \mathit{null}$. In the absence of *null* values, i.e., without null values in the original instance nor in the repair process, Definitions 7 and 8 coincide with the ones given in [3]. In [4,5,15] repairs with non *null* values have been considered.

Example 7. Consider the universal integrity constraint $\forall xy(P(x, y) \rightarrow R(x, y))$ together with the referential integrity constraint $\forall x(T(x) \rightarrow \exists y P(x, y))$ and an inconsistent minimal instance of an integration system $D = \{P(a, b), T(c)\}$. The repairs for the latter are:

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

In the first repair it can be seen that the atom $P(c, \mathit{null})$ does not propagate through the universal constraint to $R(c, \mathit{null})$. 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 $\Delta(D, D_1) \leqslant_D \Delta(D, D_7) = \{R(a, b), P(c, a)\}$.

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 7, from a minimal legal global database instance. If $\mathcal{G}$ is already consistent, then the repairs are the elements of $\mathit{Mininst}(\mathcal{G})$. In Definition 8 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 $\mathit{Linst}(\mathcal{G})$. 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 8 (*Example 1 continued*). The only element in $\mathit{Mininst}(\mathcal{G}_1)$ is $D_0 = \{R(a, b), R(c, d), R(a, c), R(d, e)\}$, that does not satisfy IC. Then, $\mathcal{G}_1$ 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 $\mathit{Linst}(\mathcal{G}_1)$.

Definition 9 [9]. (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 $\mathit{Consis}_{\mathcal{G}}(Q)$ the set of consistent answers to Q in $\mathcal{G}$.

Example 9 (*Example 8 continued*). For the query $Q_1(X)$: $\exists Y\, R(X,Y)$, the consistent answers are a, c, d. $Q_2(X,Y)$: $R(X,Y)$ has $(c,d), (d,e)$ as consistent answers.

If $\mathcal{G}$ is consistent with respect to *IC*, then $\mathit{Consis}_{\mathcal{G}}(Q) = \mathit{Minimal}_{\mathcal{G}}(Q)$. Furthermore, if the ICs are generic, then for any $\mathcal{G}$ it holds $\mathit{Consis}_{\mathcal{G}}(Q) \subseteq \mathit{Minimal}_{\mathcal{G}}(Q)$ [9]. Notice also that the notion of consistent answer can be applied to queries expressed in Datalog or its extensions with built-ins and negation.

3. Specification of minimal instances

The specification of the class $\mathit{Mininst}(\mathcal{G})$ for system $\mathcal{G}$ is given using normal deductive databases, 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 [26]. We consider global system all of whose sources are open.

3.1. The simple program

In this section we will present a first approach to the specification of legal instances. In Section 3.2 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, and already contains the key ideas.

Definition 10. Given an open global system $\mathcal{G}$, the logic program $\Pi(\mathcal{G})$, contains the following clauses:

(1) Fact $\mathit{dom}(a)$ for every constant $a \in \mathcal{U}$; and the fact $V_i(\bar{a})$ whenever $\bar{a} \in v_i$ for some source extension v_i in $\mathcal{G}$.
(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

$$P_j(\bar{X}_j) \leftarrow V_i(\bar{X}), \quad \bigwedge_{Z_l \in (\bar{X}_j \setminus \bar{X})} F_i^l(\bar{X}, Z_l), \quad j = 1, \ldots, n.$$

(3) For every predicate $F_i^l(\bar{X}, Z_l)$ introduced in (2), the rule

$$F_i^l(\bar{X}, Z_l) \leftarrow V_i(\bar{X}),\ \mathit{dom}(Z_l),\ \mathit{choice}\big((\bar{X}), (Z_l)\big).$$

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 2.1, 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 $\mathit{choice}((\bar{X}), (Z_l))$ ensures that for every (tuple of) value(s) for $\bar{X}$, only one (tuple of) value(s) for Z_l is non deterministically chosen between the constants of the active domain.

Example 10 (*Examples 4 and 5 continued*). Program $\Pi(\mathcal{G}_2)$ 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) $F_1(X,Y,Z) \leftarrow V_1(X,Y), dom(Z), choice((X,Y),(Z)).$

In this section we will restrict ourselves to a finite domain $\mathcal{U}$, what 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 6.2 we study how to handle infinite domains by adding to the active domain a finite number of extra constants, like constant u here.[4]

For every program Π with the choice operator, there is its *stable version* $SV(\Pi)$, whose stable models correspond to the so-called *choice models* of Π [26]. The program $SV(\Pi)$ is obtained as follows:

(a) Each choice rule $r: H \leftarrow B, choice((\bar{X}),(Y))$ in Π is replaced by the rule $H \leftarrow B$, $chosen_r(\bar{X},Y)$.

(b) 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 (b) 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 11 (*Example 10 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:

$$\begin{aligned}\mathcal{M}_1 = \big\{&dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a,b),\ V_2(a,c),\\ &\underline{P(a,c)},\ diffChoice_1(a,b,a),\ chosen_1(a,b,b),\ diffChoice_1(a,b,c),\\ &diffChoice_1(a,b,u),\ F_1(a,b,b),\ \underline{R(b,b)},\ \underline{P(a,b)}\big\}.\end{aligned}$$

[4] In principle, *null* could be in the domain, and then we should include $dom(null)$ among the atoms, and, since we do not want legal instances to contain the null value, the literal $Z \neq null$ in the body of the rule in (3). Instead, to keep things simpler, we will not include $dom(null)$ in $\Pi(\mathcal{G})$, even if *null* belongs to the underlying domain $\mathcal{U}$.

$$\mathcal{M}_2 = \big\{dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a,b),\ V_2(a,c),\\ \underline{P(a,c)},\ chosen_1(a,b,a),\ diffChoice_1(a,b,b),\ diffChoice_1(a,b,c),\\ diffChoice_1(a,b,u),\ F_1(a,b,a),\ \underline{R(a,b)},\ \underline{P(a,a)}\big\}.$$

$$\mathcal{M}_3 = \big\{dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a,b),\ V_2(a,c),\\ \underline{P(a,c)},\ diffChoice_1(a,b,a),\ diffChoice_1(a,b,b),\ chosen_1(a,b,c),\\ diffChoice_1(a,b,u),\ F_1(a,b,c),\ \underline{R(c,b)}\big\}.$$

$$\mathcal{M}_4 = \big\{dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a,b),\ V_2(a,c),\ \underline{P(a,c)},\\ diffChoice_1(a,b,a),\ diffChoice_1(a,b,b),\ diffChoice_1(a,b,c),\\ chosen_1(a,b,u),\ F_1(a,b,u), \underline{R(u,b)},\ \underline{P(a,u)}\big\}.$$

The underlined atoms of the models correspond to the elements in which we are interested, namely the global relations of the integration system.

Definition 11. The global instance associated to a choice model $\mathcal{M}$ of $\Pi(\mathcal{G})$ is $D_{\mathcal{M}} = \{P(\bar{a}) \mid P \in \mathcal{R}$ and $P(\bar{a}) \in \mathcal{M}\}$.

Example 12 (*Example 11 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 1. *It holds that*

$$Mininst(\mathcal{G}) \subseteq \big\{D_{\mathcal{M}} \mid \mathcal{M} \textit{ is a choice model of } \Pi(\mathcal{G})\big\} \subseteq Linst(\mathcal{G}).$$

From the inclusions in the theorem 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 4.

In Example 12 the stable models are in a one to one correspondence with the minimal legal instances, but this may not be always the case.

Example 13. 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 10 captures the minimal instances. This is important because the general program to be presented in Section 3.2 is much more complex than the simple version presented so far.

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 a 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 $S_1^0 = \{T(X, Y)\}$. *Sec* denotes the set of all view sections for system $\mathcal{G}$.

Given a view section S_i^l, we denote by $\mathit{Const}(S_i^l)$, $\mathit{UVar}(S_i^l)$ and $\mathit{EVar}(S_i^l)$ 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 $h : \mathit{Const}(S_i^l) \cup \mathit{UVar}(S_i^l) \cup \mathit{EVar}(S_i^l) \rightarrow \mathit{Const}(S_i^l) \cup \{\mu, \varepsilon\}$, such that:

(a) $h(c) = c$ for every $c \in \mathit{Const}(S_i^l)$;
(b) $h(X) = D$ with $D \in \mathit{Const}(S_i^l) \cup \{\mu\}$ for every $X \in \mathit{UVar}(S_i^l)$;
(c) $h(Z) = F$ with $F \in \mathit{Const}(S_i^l) \cup \{\mu, \varepsilon\}$ for every $Z \in \mathit{EVar}(S_i^l)$.

A particular admissible mapping L is given by

(a) $L(c) = c$ for every $c \in \mathit{Const}(S_i^l)$;
(b) $L(X) = \mu$ for every $X \in \mathit{UVar}(S_i^l)$;
(c) $L(Z) = \varepsilon$ for every $Z \in \mathit{EVar}(S_i^l)$.

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 2. *Given an integration system G, if for every view section S_i^l with existential variables, there is no admissible mapping h for S_i^l, such that $h(S_i^l) \subseteq \bigcup_{S \in (\mathit{Sec} \setminus \{S_i^l\})} L(S)$, then the instances associated to the stable models of the simple version of $\Pi(\mathcal{G})$ are exactly the minimal legal instances of $\mathcal{G}$.*

Basically, the theorem says that if there is an admissible mapping, such that $h(S_i^l) \subseteq \bigcup_{S \in (\mathit{Sec} \setminus \{S_i^l\})} L(S)$, 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 14 (*Example 13 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 that $h(S_1^Y) = \{P(\mu, \mu)\} \subseteq L(S_2^0)$. The conditions of the theorem are

Table 1

Annotation	Atom	The tuple $P(\bar{a})$ is...
$\mathbf{t_d}$	$P(\bar{a}, \mathbf{t_d})$	an atom of the minimal legal instances
$\mathbf{t_o}$	$P(\bar{a}, \mathbf{t_o})$	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

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.

Example 15 (*Examples 4 and 5 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 $h(S_2^0) \subseteq L(S_1^Z)$. In consequence, for any source contents, the simple version of $\Pi(\mathcal{G}_2)$ will calculate exactly the minimal instances of $\mathcal{G}_2$.

3.2. *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 we will introduce auxiliary annotation constants that will be used as extra arguments in the database predicates. They and their intended semantics are given in Table 1.

Definition 12. Given an open global system $\mathcal{G}$, the refined program $\Pi(\mathcal{G})$, contains the following clauses:

1. Fact $dom(a)$ for every constant $a \in \mathcal{U}$.
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,

$$P_k(\bar{X}_k, \mathbf{v_{ij}}) \leftarrow add_{v_{ij}}(\bar{X}'), \quad \bigwedge_{Z_l \in (\bar{X}_k \setminus \bar{X}')} F_i^l(\bar{X}', Z_l), \quad \text{for } P_k \in S_{ij}.$$

$$add_{v_{ij}}(\bar{X}') \leftarrow V_i(\bar{X}),\ not\, aux_{v_{ij}}(\bar{X}'), \quad \text{where } \bar{X}' = \bar{X} \cap \left\{ \bigcup_{P_k \in S_{ij}} X_k \right\}.$$

$$aux_{v_{ij}}(\bar{X}') \leftarrow \bigwedge_{l=1}^{m} var_{v_{ij} Z_l}(\bar{X}_{Z_l}).$$

$$var_{v_{ij}Z_l}(\bar{X}_{Z_l}) \leftarrow \bigwedge_{P_k \in S_{ij} \& Z_l \in \bar{X}_k} P_k(\bar{X}_k, \mathbf{nv_{ij}}),$$

$$\text{where } \bar{X}_{Z_l} = \Bigg\{ \bigcup_{P_k \in S_{ij} \& Z_l \in \bar{X}_k} X_k \Bigg\}, \quad \text{for } l = 1, \ldots, m.$$

4. For every predicate $F_i^l(\bar{X}', Z_l)$ introduced in 3(b), the rules,

$$F_i^l(\bar{X}', Z_l) \leftarrow add_{v_{ij}Z_l}(\bar{X}'),\ dom(Z_l),\ choice((\bar{X}'), (Z_l)).$$
$$add_{v_{ij}Z_l}(\bar{X}') \leftarrow add_{v_{ij}}(\bar{X}'),\ not\, aux_{v_{ij}Z_l}(\bar{X}'), \quad \text{for } l = 1, \ldots, m.$$
$$aux_{v_{ij}Z_l}(\bar{X}') \leftarrow var_{v_{ij}Z_l}(\bar{X}_{Z_l}), \bigwedge_{Z_k \neq Z_l \& Z_k \in \bar{X}_{Z_l}} F_i^k(\bar{X}', Z_k),$$

for $l = 1, \ldots, m$.

5. For every global relation $P(\bar{X})$ the rules

$$P(\bar{X}, \mathbf{nv_{ij}}) \leftarrow P(\bar{X}, \mathbf{v_{hk}}), \quad \text{for } \{(ij, hk) \mid P(\bar{X}) \in S_{ij} \cap S_{hk},\ ij \neq hk\}.$$
$$P(\bar{X}, \mathbf{nv_{ij}}) \leftarrow P(\bar{X}, \mathbf{t_o}), \quad \text{for } \{(ij) \mid P(\bar{X}) \in S_{ij}\}.$$
$$P(\bar{X}, \mathbf{t_d}) \leftarrow P(\bar{X}, \mathbf{v_{ij}}), \quad \text{for } \{(ij) \mid P(\bar{X}) \in S_{ij}\}.$$
$$P(\bar{X}, \mathbf{t_d}) \leftarrow P(\bar{X}, \mathbf{t_o}).$$

Example 16 (*Example 13 continued*). The refined program $\Pi(\mathcal{G}_3)$ is:

$$dom(a). \quad dom(c). \tag{2}$$
$$v_1(a). \quad v_2(a, c). \tag{3}$$
$$P(X, Z, \mathbf{v_1}) \leftarrow add_{v_1}(X), F_z(X, Z). \tag{4}$$
$$add_{v_1}(X) \leftarrow v_1(X),\ not\, aux_{v_1}(X). \tag{5}$$
$$aux_{v_1}(X) \leftarrow var_{v_1z}(X, Z). \tag{6}$$
$$var_{v_1z}(X, Z) \leftarrow P(X, Z, nv_1). \tag{7}$$
$$F_z(X, Z) \leftarrow add_{v_1}(X), dom(Z), chosen_{v_1z}(X, Z). \tag{8}$$
$$chosen_{v_1z}(X, Z) \leftarrow add_{v_1}(X), dom(Z),\ not\, diffChoice_{v_1z}(X, Z). \tag{9}$$
$$diffChoice_{v_1z}(X, Z) \leftarrow chosen_{v_1z}(X, Z'), dom(Z), Z' \neq Z. \tag{10}$$
$$P(X, Y, \mathbf{t_o}) \leftarrow v_2(X, Y). \tag{11}$$
$$P(X, Y, \mathbf{nv_1}) \leftarrow P(X, Y, \mathbf{t_o}). \tag{12}$$
$$P(X, Y, \mathbf{t_d}) \leftarrow P(X, Y, \mathbf{v_1}). \tag{13}$$
$$P(X, Y, \mathbf{t_d}) \leftarrow P(X, Y, \mathbf{t_o}). \tag{14}$$

Rules (4), to (7) 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 rule (4) with a Z value given by the function predicate $F_z(\bar{a}, Z)$. This function predicate is calculated by rules (8) to (10). Rule (11) enforces the satisfaction of the openness of V_2 by adding obligatory atoms to predicate P and rule (12) 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 with annotation $\mathbf{t_d}$ the elements that were generated by both views and that are in the minimal legal instances. The stable model of this program is $\{dom(a), dom(c), v_1(a), v_2(a,c), \underline{P(a,c,\mathbf{t_d})}, P(a,c,\mathbf{t_o}), P(a,c,\mathbf{nv_1}), aux_{v_1}(a)\}$, which corresponds to the only minimal legal instance $\{P(a,c)\}$.

Theorem 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 $P(\bar{a}, \mathbf{t_d}) \in \mathcal{M}\} \in Mininst(\mathcal{G})$. Furthermore, the minimal legal instances obtained in this way are all the minimal legal instances of $\mathcal{G}$.*

The 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$^{\neg}$ 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., a Datalog queries, can be immediately obtained from Theorem 3 and the fact that for those queries $Certain_{\mathcal{G}}(Q) = Minimal_{\mathcal{G}}(Q)$.

Corollary 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 $\mathcal{G}$.*

We know that under the hypothesis of Theorem 2, the simple and refined programs compute the same legal database instances, namely the minimal ones. Beyond this, it is worth mentioning that, under the same hypothesis, there is a simple mechanical, syntactic transformation of the refined program into a simple program (in the sense of Section 3.1) that has the same stable models, and then, in particular, produces the same database instances (see Appendix A.2).

4. Specification of repairs of a global system

In [6], repairs of single relational databases are specified as stable models of disjunctive logic programs. We briefly explain those programs, because they will be used to specify repairs of instances of integration systems.

First, the database predicates are expanded with an extra argument to be filled with one of a set of new annotation constants. An atom inside (outside) the original database is annotated with $\mathbf{t_d}$ ($\mathbf{f_d}$).[5] Annotations $\mathbf{t_a}$ and $\mathbf{f_a}$ are considered *advisory* values, to solve conflicts between the database and the ICs. If an atom gets the derived annotation $\mathbf{f_a}$, it means an advise to make it false, i.e., to delete it from the database. Similarly, an atom that gets the annotation $\mathbf{t_a}$, this is seen as an advice to insert it into the database.

[5] The annotation $\mathbf{t_d}$ is the same we had in the previous section, actually the program there will provide the contents of the minimal instances in terms of $\mathbf{t_d}$; next, in the repair process, the new annotations introduced here will be generated.

Example 17 (*Example 7 continued*). Consider the ICs $\forall x(P(x,y) \rightarrow R(x,y))$ and $\forall x(T(x) \rightarrow \exists y P(x,y))$, together with the inconsistent database instance $D = \{P(a,b), T(c)\}$ and a domain $\mathcal{U} = \{a,b,c,u\}$. The logic program should have the effect of repairing the database. Single, local repair steps are obtained by deriving the annotations $\mathbf{t_a}$ or $\mathbf{f_a}$. This is done when each IC is considered in isolation, but there may be interacting ICs, and the repair process may take several steps and should stabilize at some point. In order to achieve this, we use annotations $\mathbf{t}^\star$, $\mathbf{f}^\star$. The latter, for example, groups together the annotations $\mathbf{f_d}$ and $\mathbf{f_a}$ for the same atom (rules (2) and (5) below). These derived annotations are used to give a feedback to the bodies of the rules that produce the local, single repair steps, so that a propagation of changes is triggered (rule (3) below).

The annotations $\mathbf{t}^{\star\star}$ and $\mathbf{f}^{\star\star}$ are just used to read off the literals that are inside (resp. outside) a repair. This is achieved by means of rules (7) below, that are used to interpret the models as database repairs. The facts of rule (1) correspond to all the elements of the domain except for the *null* constant, which is left outside of *dom*. The following is the program:

(1) $dom(a).\quad dom(b).\quad dom(c).\quad dom(u).$

(2) $P(x,y,\mathbf{f}^\star) \leftarrow P(x,y,\mathbf{f_a}), dom(x), dom(y).$

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

$P(x,y,\mathbf{t}^\star) \leftarrow P(x,y,\mathbf{t_d}), dom(x), dom(y).$ (similarly for R and T)

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

$T(x,\mathbf{f_a}) \vee P(x,null,\mathbf{t_a}) \leftarrow T(x,\mathbf{t}^\star),\ not\, aux(x),\ not\, P(x,null,\mathbf{t_d}), dom(x).$

$aux(x) \leftarrow P(x,y,\mathbf{t_d}),\ not\, P(x,y,\mathbf{f_a}).$

$aux(x) \leftarrow P(x',y,\mathbf{t_a}).$

(4) $P(a,\mathbf{t_d}) \leftarrow .$

(5) $P(x,y,\mathbf{f}^\star) \leftarrow dom(x), dom(y),\ not\, P(x,y,\mathbf{t_d}).$ (similarly for R and T)

(6) $\leftarrow P(\bar{x},\mathbf{t_a}), P(\bar{x},\mathbf{f_a}).\quad \leftarrow R(\bar{x},\mathbf{t_a}), R(\bar{x},\mathbf{f_a}).$

(7) $P(x,y,\mathbf{t}^{\star\star}) \leftarrow P(x,y,\mathbf{t_a}), dom(x), dom(y).$

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

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

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

(similarly for R and T)

Only rules (3) depend on the ICs. The first rule in (3) corresponds to the universal ICs and the rest to the referential IC. These rules say how to repair the inconsistencies. Rules (4) contain the database atoms. Rules (5) capture the *closed world assumption* (CWA) [40]. Rules (6) are denial program constraints to discard models that contain an atom annotated with both $\mathbf{t_a}$ and $\mathbf{f_a}$. The program has four stable models. The repairs are obtained from them by selecting the atoms annotated with $\mathbf{t}^{\star\star}$: $D_1 = \{P(a,b), R(a,b)\}$, $D_2 = \{P(a,b), R(a,b), T(c), P(c,null)\}$ and $D_3 = \{T(c), P(c,null)\}$, $D_4 = \emptyset$. As expected, they coincide with the ones obtained in Example 7.

It can be proved [6] in the context of single relational databases that the stable models of these disjunctive programs are in a one to one correspondence with the repairs of the original database, for any combination of universal and acyclic referential integrity constraints. If there are cycles between the referential ICs, then the specification programs may produce a class of stable models that properly extends the class of repairs [12]. Those models that do not correspond to repairs still satisfy the ICs, but may not be minimal repairs. In this case the stable models that do not correspond to (minimal) repairs can be pruned by comparison with the other stable models [12]. These properties will be inherited by our application of this kind of programs to the specification of the repairs of the minimal instances of an integration system.

The next definition combines into one program the refined version that specifies the minimal legal instances and the specification of the repairs of those minimal instances.

Definition 13. The *repair program*, $\Pi(\mathcal{G}, IC)$, of $\mathcal{G}$ with respect to IC contains the following clauses:

(1) The same rules as in Definition 12.

(2) For every predicate $P \in \mathcal{R}$, the clauses

$$P(\bar{X}, \mathbf{t^\star}) \leftarrow P(\bar{X}, \mathbf{t_d}), dom(\bar{X}).^{6}$$
$$P(\bar{X}, \mathbf{t^\star}) \leftarrow P(\bar{X}, \mathbf{t_a}), dom(\bar{X}).$$
$$P(\bar{X}, \mathbf{f^\star}) \leftarrow P(\bar{X}, \mathbf{f_a}), dom(\bar{X}).$$
$$P(\bar{X}, \mathbf{f^\star}) \leftarrow dom(\bar{X}),\ not\, P(\bar{X}, \mathbf{t_d}).$$

(3) For every first-order global universal IC of the form $\bar{\forall}(Q_1(\bar{Y}_1) \vee \cdots \vee Q_n(\bar{Y}_n) \leftarrow P_1(\bar{X}_1) \wedge \cdots \wedge P_m(\bar{X}_m) \wedge \varphi)$, where $P_i, Q_j \in \mathcal{R}$, and φ is a conjunction of built-in atoms, the clause:

$$\bigvee_{i=1}^{n} P_i(\bar{X}_i, \mathbf{f_a}) \bigvee_{j=1}^{m} Q_j(\bar{Y}_j, \mathbf{t_a}) \leftarrow \bigwedge_{i=1}^{n} P_i(\bar{X}_i, \mathbf{t^\star}), \bigwedge_{j=1}^{m} Q_j(\bar{Y}_j, \mathbf{f^\star}), dom(\bar{X}), \varphi;$$

where $\bar{X}$ is the tuple of all variables appearing in database atoms in the rule.

(4) For every referential IC of the form $\forall \bar{X}(P(\bar{X}) \rightarrow \exists Y\, Q(\bar{X}', Y))$, with $\bar{X}' \subseteq \bar{X}$, the clauses

$$P(\bar{X}, \mathbf{f_a}) \vee Q(\bar{X}', null, \mathbf{t_a}) \leftarrow P(\bar{X}, \mathbf{t^\star}),\ not\, aux(\bar{X}'),\ not\, Q(\bar{X}', null, \mathbf{t_d}),\ dom(\bar{X}).$$
$$aux(\bar{X}') \leftarrow Q(\bar{X}', Y, \mathbf{t_d}), not\, Q(\bar{X}', Y, \mathbf{f_a}), dom(\bar{X}', Y).$$
$$aux(\bar{X}') \leftarrow Q(\bar{X}', Y, \mathbf{t_a}), dom(\bar{X}', Y).$$

(5) For every predicate $P \in \mathcal{R}$, the interpretation clauses:

$$P(\bar{a}, \mathbf{f^{\star\star}}) \leftarrow P(\bar{a}, \mathbf{f_a}).$$

[6] If $\bar{X} = (X_1, \ldots, X_n)$, we abbreviate $dom(X_1) \wedge \cdots \wedge dom(X_n)$ with $dom(\bar{X})$.

$$P(\bar{a}, \mathbf{f}^{\star\star}) \leftarrow \mathit{not}\ P(\bar{a}, \mathbf{t_d}),\ \mathit{not}\ P(\bar{a}, \mathbf{t_a}).$$
$$P(\bar{a}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{a}, \mathbf{t_a}).$$
$$P(\bar{a}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{a}, \mathbf{t_d}),\ \mathit{not}\ P(\bar{a}, \mathbf{f_a}).$$

Rules (4) repair referential ICs by deletion of tuples or insertion of null values that are not propagated through other ICs [6]. For this purpose, *dom*(*null*) is not considered as a fact and therefore the *null* values will not propagate. Optimizations of the repair part of the program, like avoiding the materialization of the CWA, are analyzed in [6].

The choice models of program $\Pi(\mathcal{G}, IC)$ that do not contain a pair of literals of the form $\{P(\bar{a}, \mathbf{t_a}), P(\bar{a}, \mathbf{f_a})\}$ are called *coherent models*. Only coherent models can be obtained for the program if the denial constraints of the form $\leftarrow P(\bar{x}, \mathbf{t}^{\star\star}), P(\bar{x}, \mathbf{f}^{\star\star})$ are included in the program.

Definition 14. The global instance associated to a choice model $\mathcal{M}$ of $\Pi(\mathcal{G}, IC)$ is $D_{\mathcal{M}} = \{P(\bar{a}) \mid P \in \mathcal{R}$ and $P(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}\}$.

The repair program can be split [37] into the specification of the minimal instances and the specification of their repairs. Therefore, the minimal legal instances can be calculated first, and then the repairs of them. Each minimal model calculated 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 straightforwardly.

Theorem 4. *Let IC be an arbitrary class of universal and acyclic referential integrity constraints. If $\mathcal{M}$ is a coherent 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.*

In the case in which a cyclic set of referential ICs is considered, the global instances associated to the choice models of the program will 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 minimally differ from the minimal legal instance [12].

5. Consistent answers

Now we can obtain the answers to queries posed to a system $\mathcal{G}$ that are consistent with respect to *IC*. First we will consider universal and acyclic referential ICs. We do the following:

(1) We start with a query Q that is expressed, e.g., as a stratified Datalog program, $\Pi(Q)$, whose extensional predicates are elements of the global schema $\mathcal{R}$. Each positive occurrence of those predicates, say $P(\bar{t})$, is replaced by $P(\bar{t}, \mathbf{t}^{\star\star})$; and each negative occurrence, say not $P(\bar{t})$, by $P(\bar{t}, \mathbf{f}^{\star\star})$. This query program has a query predicate *Ans* that collects the answers to Q. In particular, first order queries can be expressed as stratified Datalog programs [1].

(2) Program $\Pi(Q)$ is appended to the program $SV(\Pi(\mathcal{G}, IC))$, the stable version of the repair program.
(3) The consistent answers to Q are the ground *Ans* atoms in the intersection of all stable models of $\Pi(Q) \cup SV(\Pi(\mathcal{G}, IC))$.

Example 18 (*Example 11 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 $\mathcal{G}_2$. We want the consistent answers to the query $Q : P(x, y)$. First, the query is written as the query program clause $Ans(X, Y) \leftarrow P(X, Y, \mathbf{t}^{\star\star})$. This query program, $\Pi(Q)$, is run with the revised version of $SV(\Pi(\mathcal{G}_3, sim))$ that has the following rules:

% Subprogram for minimal instances

$dom(a). \quad dom(b). \quad dom(c). \quad dom(u).$

$v_1(a, b). \quad v_2(a, c).$

$P(X, Y, \mathbf{nv_1}) \leftarrow P(X, Y, \mathbf{t_o}).$

$P(X, Y, \mathbf{nv_2}) \leftarrow P(X, Y, \mathbf{v_1}).$

$P(X, Y, \mathbf{t_d}) \leftarrow P(X, Y, \mathbf{v_1}).$

$P(X, Y, \mathbf{t_d}) \leftarrow P(X, Y, \mathbf{t_o}).$

$R(X, Y, \mathbf{t_d}) \leftarrow R(X, Y, \mathbf{v_1}).$

% Specification of V_1

$P(X, Y, \mathbf{v_1}) \leftarrow add_{v_1}(X, Z), F_1^Y(X, Z, Y).$

$R(Y, Z, \mathbf{v_1}) \leftarrow add_{v_1}(X, Z), F_1^Y(X, Z, Y).$

$add_{v_1}(X, Z) \leftarrow v_1(X, Z),\ not\, aux_{v_1}(X, Z).$

$aux_{v_1}(X, Z) \leftarrow var_{v_1 Y}(X, Y, Z).$

$var_{v_1 Y}(X, Y, Z) \leftarrow P(X, Y, \mathbf{nv_1}), R(Y, Z, \mathbf{nv_1}).$

$F_1^Y(X, Z, Y) \leftarrow add_{v_1 Y}(X, Z), dom(Y), chosen_{v_1 Y}(X, Z, Y).$

$chosen_{v_1 Y}(X, Z, Y) \leftarrow add_{v_1 Y}(X, Z), dom(Y),\ not\, diffchoice_{v_1}(X, Z, Y).$

$diffchoice_{v_1}(X, Z, Y) \leftarrow chosen_{v_1 Y}(X, Z, Y'), dom(Y), Y' \neq Y.$

$add_{v_1 Y}(X, Z) \leftarrow add_{v_1}(X, Z),\ not\, aux_{v_1 Y}(X, Z).$

$aux_{v_1 Y}(X, Z) \leftarrow var_{v_1 Y}(X, Y, Z).$

% Specification of V_2

$P(X, Y, \mathbf{t_o}) \leftarrow v_2(X, Y).$

% Repair subprogram

$P(X, Y, \mathbf{t}^\star) \leftarrow P(X, Y, \mathbf{t_a}), dom(X), dom(Y).$

$P(X, Y, \mathbf{t}^\star) \leftarrow P(X, Y, \mathbf{t_d}), dom(X), dom(Y).$

$$
\begin{aligned}
&P(X,Y,\mathbf{f^\star}) \leftarrow dom(X), dom(Y),\ not\ P(X,Y,\mathbf{t_d}).\\
&P(X,Y,\mathbf{f^\star}) \leftarrow P(X,Y,\mathbf{f_a}), dom(X), dom(Y).\\
&R(X,Y,\mathbf{t^\star}) \leftarrow R(X,Y,\mathbf{t_a}), dom(X), dom(Y).\\
&R(X,Y,\mathbf{t^\star}) \leftarrow R(X,Y,\mathbf{t_d}), dom(X), dom(Y).\\
&R(X,Y,\mathbf{f^\star}) \leftarrow dom(X), dom(Y),\ not\ R(X,Y,\mathbf{t_d}).\\
&R(X,Y,\mathbf{f^\star}) \leftarrow R(X,Y,\mathbf{f_a}), dom(X), dom(Y).\\
&R(X,Y,\mathbf{f_a}) \vee R(Y,X,\mathbf{t_a}) \leftarrow R(X,Y,\mathbf{t^\star}), R(Y,X,\mathbf{f^\star}), dom(X), dom(Y).\\
&P(X,Y,\mathbf{t^{\star\star}}) \leftarrow P(X,Y,\mathbf{t_a}), dom(X), dom(Y).\\
&P(X,Y,\mathbf{t^{\star\star}}) \leftarrow P(X,Y,\mathbf{t_d}), dom(X), dom(Y),\ not\ P(X,Y,\mathbf{f_a}).\\
&P(X,Y,\mathbf{f^{\star\star}}) \leftarrow P(X,Y,\mathbf{f_a}), dom(X), dom(Y).\\
&P(X,Y,\mathbf{f^{\star\star}}) \leftarrow dom(X), dom(Y),\ not\ P(X,Y,\mathbf{t_d}),\ not\ P(X,Y,\mathbf{t_a}).\\
&R(X,Y,\mathbf{t^{\star\star}}) \leftarrow R(X,Y,\mathbf{t_a}), dom(X), dom(Y).\\
&R(X,Y,\mathbf{t^{\star\star}}) \leftarrow R(X,Y,\mathbf{t_d}), dom(X), dom(Y),\ not\ R(X,Y,\mathbf{f_a}).\\
&R(X,Y,\mathbf{f^{\star\star}}) \leftarrow R(X,Y,\mathbf{f_a}), dom(X), dom(Y).\\
&R(X,Y,\mathbf{f^{\star\star}}) \leftarrow dom(X), dom(Y),\ not\ R(X,Y,\mathbf{t_d}),\ not\ R(X,Y,\mathbf{t_a}).\\
&\qquad\qquad\quad \leftarrow R(X,Y,\mathbf{t_a}), R(X,Y,\mathbf{f_a}).\\
&\qquad\qquad\quad \leftarrow P(X,Y,\mathbf{t_a}), P(X,Y,\mathbf{f_a}).
\end{aligned}
$$

This program 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 12; (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 $D_{\mathcal{M}_4}$.

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. Further analysis, extensions and discussion

6.1. Complexity

The complexity analysis of consistent query answering in integration of open sources under the LAV approach 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 minimal 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 [41]). Actually, if the choice operator program is represented in its "classical" stable version (see Section 3.1), 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 [19,35] in data complexity [1], in our case, 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 [2]. With disjunctive views, as considered in Section 6.4, the complexity of the program goes up to being 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 [19], which matches the complexity of consistent query answering [10,15,18].

There are some cases studied in [6], e.g., only universal ICs, where the repair part of the program for CQA is *head-cycle free* (HCF) and therefore the complexity is reduced to coNP [7,34]. This coNP-completeness result can be extended to some cases where both universal and RICs are considered. It is possible to show [12] that 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. In [15,18] some cases where functional dependencies and referential integrities coexist are presented, for which the problem of CQA becomes Π_2^P-complete. Actually, in the case when repairs with respect to cyclic RICs is done by introducing arbitrary, non null elements of the underlying domain, the problem of consistent query answering becomes undecidable [15]. However, if repairs with respect to cyclic RICs are obtained by introducing null values that do not propagate via ICs, the problem of consistent query answering becomes decidable [12].

6.2. Infinite vs. finite domain

In Section 2.1 we considered the possibility of having an infinite underlying domain $\mathcal{U}$. At the purely specification level there is not problem in admitting, in the first item of Definition 10, 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, cf. Example 10 (the finite domain is specified by the *dom* predicate), but also an extra constant u that does not appear in the active domain of the integration system, that consists of all the constants in the sources plus those that appear in the view definitions. The reason is that we need a finite domain to run the programs, 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 19. 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 $\{(a)\}$, we have that the active domain is $\{a\}$ and that $R(a, a)$ is in all the legal instances of $\mathcal{G}_4$ if only this domain is used; and we would have $\mathit{Certain}_{\mathcal{G}_4}(Q) = \{a\}$. 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.

In Example 10, 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 underlying domain, as we saw in Example 5, where $\mathit{Mininst}(\mathcal{G}_2) = \{\{P(a, c), P(a, z), R(z, a)\} \mid z \in \mathcal{U} = \{a, b, c, \ldots\}\}$.

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 19). In consequence, we need a larger domain, with enough variables to represent the relations 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 10).

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 20. Given 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 good enough to add to the active domain $\{a, b, c, d\}$ three extra constants, 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.

6.3. Choice models vs. Skolem functions

In this paper 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, which was one of our original goals, is interesting in itself, and 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 it, and then filtering out the final answers that contain those functions (as done in [21]).

We first analyze the case of the simple program (see Section 3.1), in which we want to consider using the 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 couple of rules $P(X,Y) \leftarrow V(X), F(X,Y)$ and $F(X,Y) \leftarrow V(X), \mathit{dom}(Y), \mathit{choice}((X),(Y))$.

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 to the program with Skolem functions, the answer set may contain or not answers with 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 6.2) 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. In consequence, for computing the certain answers to a monotone query, we can indistinctly use 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 3.2). 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 21. Consider an integration system $\mathcal{G}$ with

$$V_1(X) \leftarrow P(X_1,Y_1,Z_1), S(Y_1), \quad V_1(a),$$
$$V_2(X,Y) \leftarrow P(X_2,Y_2,Z_2), \qquad V_2(a,e).$$

The following is the program with Skolem functions:

$$\%V_1$$
$$P(X, f_1(X), f_2(X), \mathbf{v_1}) \leftarrow \mathit{add}_{v_1}(X), \mathit{add}_{v_1Y}(X), \mathit{add}_{v_1Z}(X).$$
$$S(f_1(X), \mathbf{v_1}) \leftarrow \mathit{add}_{v_1}(X).$$
$$\mathit{add}_{v_1}(X) \leftarrow v_1(X),\ \mathit{not}\, \mathit{aux}_{v_1}(X).$$
$$\mathit{aux}_{v_1}(X) \leftarrow \mathit{var}_{v_1Y}(X,Y,Z), \mathit{var}_{v_1Z}(X,Y,Z).$$

$var_{v_1Y}(X, Y, Z) \leftarrow P(X, Y, Z, \mathbf{nv_1}), S(Y, \mathbf{nv_1}).$

$var_{v_1Z}(X, Y, Z) \leftarrow P(X, Y, Z, \mathbf{nv_1}).$

$add_{v_1Y}(X) \leftarrow add_{v_1}(X),\ not\, aux_{v_1Y}(X).$

$aux_{v_1Y}(X) \leftarrow var_{v_1Y}(X, Y, Z), Z = f_2(X).$

$add_{v_1Z}(X) \leftarrow add_{v_1}(X),\ not\, aux_{v_1Z}(X).$

$aux_{v_1Z}(X) \leftarrow var_{v_1Z}(X, Y, Z), Z = f_1(X).$

$\%V_2$

$P(X, Y, f_3(X, Y), \mathbf{v_2}) \leftarrow add_{v_2}(X, Y), add_{v_2Z}(X, Y).$

$add_{v_2}(X, Y) \leftarrow v_2(X, Y),\ not\, aux_{v_2}(X, Y).$

$aux_{v_2}(X, Y) \leftarrow var_{v_2Z}(X, Y, Z).$

$var_{v_2Z}(X, Y, Z) \leftarrow P(X, Y, Z, \mathbf{nv_2}).$

$add_{v_2Z}(X, Y) \leftarrow add_{v_2}(X, Y),\ not\, aux_{v_2Z}(X, Y).$

$aux_{v_2Z}(X, Y) \leftarrow var_{v_2Z}(X, Y, Z).$

$P(X, Y, Z, \mathbf{nv_1}) \leftarrow P(X, Y, Z, \mathbf{v_2}).$

$P(X, Y, Z, \mathbf{nv_2}) \leftarrow P(X, Y, Z, \mathbf{v_1}).$

$P(X, Y, Z, \mathbf{t_d}) \leftarrow P(X, Y, Z, \mathbf{v_1}).$

$P(X, Y, Z, \mathbf{t_d}) \leftarrow P(X, Y, Z, \mathbf{v_2}).$

$S(Y, \mathbf{t_d}) \leftarrow S(Y, \mathbf{v_1}).$

The stable models of the refined program with Skolem functions are calculated under the *unique names assumption* [40]. 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 3.1). For example, if two atoms, say $P(a, f1(a), f2(a))$ and $P(a, e, f3(a, e))$, 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, what 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 22 (*Example 21 continued*). The minimal instances of this integration system can be represented by $\{\{P(a, e, f_3(a, e)), P(a, f_1(a), f_2(a)), S(f_1(a))\} \mid f_3(a, e) \in \mathcal{U}, f_2(a) \in \mathcal{U}, f_1(a) \in \mathcal{U}\backslash\{e\}\} \cup \{\{P(a, e, f_3(a, e)), S(e)\} \mid f_3(a, e) \in \mathcal{U}\}$. 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 $P(X, f_1(X), f_2(X), v_1)$. In 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 23 (*Examples 4 and 5 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 by the simple program with Skolem functions. This model is inconsistent with respect to $IC : \forall x \forall y (R(X, Y) \rightarrow R(Y, X))$.

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 set of 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.

Additional remarks on this issue can be found in [8].

6.4. Disjunctive sources

In Section 3 we considered sources defined as conjunctive views only. If sources are now described as disjunctive views, i.e., with more than one conjunctive rule [20], 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 $\varphi_{i1}, \ldots, \varphi_{im}$, and v_i is the given extension of the source.

Definition 15. Given an open global system $\mathcal{G} = \{\langle \Phi_1, v_1 \rangle, \ldots, \langle \Phi_n, v_n \rangle\}$, the set of legal global instances is $Linst(\mathcal{G}) = \{D$ instance over $\mathcal{R} \mid v_i \subseteq \bigcup_k \varphi_{ik}(D),$ for $i = 1, \ldots, n\}$.

Example 24. 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 following view definitions and extensions:

Source	Extension	View definitions
v_1	$\{V_1(a,b), V_1(c,d)\}$	$\mathcal{V}_{11}: V_1(X,Y) \leftarrow R(X,Y), S(Y)$
		$\mathcal{V}_{12}: V_1(X,d) \leftarrow T(X,d)$
v_2	$\{V_2(b), V_2(a)\}$	$\mathcal{V}_{21}: V_2(X) \leftarrow S(X)$

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 $\{S(b), S(a), R(a,b), T(c,d), T(a,b)\}$.

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 [20] 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 25 (*Example 24 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 not 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 by the following table:

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

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 25, the following rule needs to be imposed

$$\big(R(X,d) \wedge S(d)\big) \vee T(X,d) \leftarrow V(X,d), \tag{15}$$

which is equivalent to the pair of disjunctive Datalog rules

$$R(X,d) \vee T(X,d) \leftarrow V(X,d), \tag{16}$$

$$S(d) \vee T(X,d) \leftarrow V(X,d). \tag{17}$$

For each set of truly disjunctive views, rules like (16) and (17) will have to be satisfied by the legal instances. These remarks motivate the following program as an specification of the minimal legal instances.

Definition 16. Given an open global system $\mathcal{G}$, the program, $\Pi^{\vee}(\mathcal{G})$, contains the following clauses:

(1) Fact $dom(a)$ for every constant $a \in \mathcal{U}$; and 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

$$\mathcal{V}_{i1}: V_i(\bar{X}_1) \leftarrow P_{11}(\bar{X}_{11}), \ldots, P_{1n}(\bar{X}_{1n_1}),$$
$$\ldots$$
$$\mathcal{V}_{ik}: V_i(\bar{X}_k) \leftarrow P_{k1}(\bar{X}_{k1}), \ldots, P_{kn}(\bar{X}_{kn_k}),$$

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

$$P_{1\delta_1}(\bar{X}'_{1\delta_1}) \vee \cdots \vee P_{k\delta_k}(\bar{X}'_{k\delta_k}) \leftarrow V_i(\bar{W}) \wedge \varphi \wedge \bigwedge_{Z_l \in (\bar{X}' \setminus \bar{W})} F_i^l(\bar{W}, Z_l),$$

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 vectors $\bar{X}'_{1\delta_1}, \ldots, \bar{X}'_{k\delta_k}$ 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

$$F_i^l(\bar{X}, Z_l) \leftarrow V_i(\bar{X}), dom(Z_l), choice((\bar{X}), (Z_l)).$$

Example 26 (*Example 25 continued*). The program $\Pi^{\vee}(\mathcal{G}_7)$ is:

$$dom(a). \quad dom(b). \quad dom(c). \quad dom(d). \tag{18}$$
$$R(X, Y) \leftarrow V_1(X, Y), Y \neq d. \tag{19}$$
$$S(Y) \leftarrow V_1(X, Y), Y \neq d. \tag{20}$$
$$T(X, d) \vee R(X, Y) \leftarrow V_1(X, Y). \tag{21}$$
$$T(X, d) \vee S(Y) \leftarrow V_1(X, Y). \tag{22}$$
$$S(X) \leftarrow V_2(X). \tag{23}$$

Rules (19)–(20) and (21)–(22) represent, respectively, the first and second truly disjunctive set for source v_1. Rule (23) is for the non-disjunctive source v_2.

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 3.1. 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, corresponding to the non-disjunctive program in Section 3.2, for which $Mininst(\mathcal{G}) = \{D_{\mathcal{M}} \mid \mathcal{M}$ is a stable model of $\Pi^{\vee}(\mathcal{G})\}$ also holds.

7. Conclusions

We have presented a general approach to specifying, by means of disjunctive deductive databases with stable model semantics, the database repairs of a mediated integration system with open sources under the LAV approach. Then, 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 of them. With respect to 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 [9] for integration systems, and the one in [6] 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.

For reasons of space, we just mention a few optimizations of the specification programs and their execution (more on optimization of repair programs can be found in [6]). The materialization of the CWA present in $\Pi(\mathcal{G}, IC)$ can be avoided by program transformation. We have identified classes of common ICs for which $SV(\Pi(\mathcal{G}, IC))$ becomes head-cycle-free, and in consequence, can be transformed into a non-disjunctive program [7,34]. Transformations are shown in [6].

The program for CQA can be split [37] into: (1) the program that specifies minimal legal instances; (2) the program that specifies their repairs; and (3) the query program. If the simple version can be used in (1), that layer is a stratified program. Otherwise, if the refined version is used, that layer is not stratified, but its models can be computed bottom-up as fixpoints of an iterative operator [27]. The second layer, i.e., the repair part, is *locally stratified* [39]. Finally, if the query program is stratified, e.g., if the original query is first-order, then the consistent answers can be eventually computed by a bottom-up evaluation mechanism.

We have already indicated that in the case the set of ICs contain referential ICs with cycles between them the stable models of the specification programs we gave may correspond to a superclass of the repairs of the global system [12]. Non minimal repairs may appear as models of the program. It should be possible to modify the given program by adding a new layer of rules that does the job of pruning all the stable models of the original program that do not correspond to (minimal) repairs. In this direction the answer set programming based specification of some "local test" for minimality as given in [38] (and used in [11] in the context of database repairs) could be attempted.

For CQA from integration systems we have successfully experimented with *DLV* [22, 35]. The current implementations of the disjunctive stable models semantics would be much more effective in database applications if it were possible to evaluate open queries in a form that is guided by the query rather than based on, first, massive grounding of the whole program and, second, considering what can be found in every (completely constructed) stable model of the program. First optimizations of this kind have been reported in [23].

With respect to related papers, query answering in mediated integration systems *under the assumption* that certain global ICs hold has been treated in [14,21,29,31]. However, in CQA, we do not assume that global ICs hold. Logic programming specifications of repairs of single relational databases have been presented in [4,5,30].

In [9], 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 [3] 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$^{\neg}$ queries. This approach is limited by the restrictions of the query transformation methodology. In particular, it can be applied only to queries that are conjunctions of literals and universal ICs.

Integration systems under the GAV approach that do not satisfy global key dependencies are considered in [32]. 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 [16] 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 [8].

With respect to current and future work, apart from considering all kinds of implementation and optimization issues around the programs and their interaction with a database, we have extended [8] our treatment of CQA in integration systems to the mixed case where open, closed and sources that are both open and closed (clopen) coexist [28]; and to particular, but common and natural combinations of them. We are working on identifying conditions on the view definitions that make it possible to compute, from the program $\Pi(\mathcal{G})$, the certain answers to possibly non-monotonic queries.

In this paper we have considered null values based repairs under RICs. The null values have a special treatment with respect to satisfaction of ICs, and as a consequence, they do not propagate in the repair process. In [4,5,15], repairs of RICs using normal domain values are considered. This, under cyclic sets of RICs, may lead to undecidability of consistent query answering. It would be interesting to study some sort of mixed approach, and also the possibility of limited propagation of null values.

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.

Acknowledgements

Research funded by DIPUC, CONICYT, Carleton University Start-Up Grant 9364-01, NSERC Grant 250279-02, and a grant from the CITO/IBM-CAS Student Internship Program. L. Bertossi is Faculty Fellow of the IBM Center for Advanced Studies, Toronto Lab. We also appreciate a CoLogNet Scholarship for Loreto Bravo to attend the Workshop on Logic-Based Method for Information Integration (Vienna, August 2003). We are grateful to Alberto Mendelzon, Pablo Barceló, Jan Chomicki, Enrico Franconi, Andrei Lopatenko for useful conversations, and to the anonymous referees for [13], a first version of this paper, for useful remarks.

Appendix A

A.1. Proof of results

Proof of Theorem 1. Consider $\Pi(G)$ as in Definition 10. First we prove:

$$\{D_{\mathcal{M}} \mid \mathcal{M} \text{ is a choice model of } \Pi(\mathcal{G})\} \subseteq \mathit{Linst}(\mathcal{G}). \tag{A.1}$$

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 $v_i \nsubseteq \varphi_i(D_{\mathcal{M}})$, 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 $\neg\exists\bar{z}(P_1(\bar{a}_1, \bar{z}_1) \wedge \cdots \wedge P_n(\bar{a}_n, \bar{z}_n))$, for $\bar{a}_i \subseteq \bar{a}$, and $\bar{z}_i \subseteq \bar{z}$. This is equivalent to

$$\forall\bar{z}\big(\neg P_1(\bar{a}_1, \bar{z}_1) \vee \cdots \vee \neg P_n(\bar{a}_n, \bar{z}_n)\big). \tag{A.2}$$

A consequence of (A.2) and rules (2) of $\Pi(\mathcal{G})$ is the following:

$$\forall\bar{z}\left(\neg V_i(\bar{a}) \vee \bigvee_l \neg F_i^l(\bar{a}, z_l)\right). \tag{A.3}$$

Since $V_i(\bar{a}) \in \mathcal{M}$ and rules (3) of $\Pi(G)$ are satisfied by $\mathcal{M}$ we have that for some b's in the domain the atoms $F_i^l(\bar{a}, b) \in \mathcal{M}$. But we had that Eq. (A.3) holds. We have reached a contradiction because (A.3) is false in $\mathcal{M}$; and (A.1) is proven.

Now we want to prove: $\mathit{Mininst}(\mathcal{G}) \subseteq \{D_{\mathcal{M}} \mid \mathcal{M}$ is a choice model of $\Pi(\mathcal{G})\}$.

The program $\Pi(\mathcal{G})$ can be split [37] 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 [41]. 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) $\mathit{dom}(a)$ for every constant $a \in \mathcal{U}$;
(4) For each view $V_i(\bar{x})$, consider the rules $F_i^l(\bar{x}, z_l) \leftarrow \mathit{body}(\varphi_{V_i})$, 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 $\mathit{choice}(\bar{a}, b)$ 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 $\mathcal{M}_B := (\mathcal{M} \backslash D)$ 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 $b \in \mathcal{U}$ for each z_l 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 $D: F_i^l(\bar{a}, b_l) \leftarrow \mathit{body}(\varphi_{V_i})$ and $F_i^l(\bar{a}, b_2) \leftarrow \mathit{body}(\varphi_{V_i})$. This would imply that D has two set of atoms satisfying the mapping $V_i(\bar{a}) \leftarrow \mathit{body}(\varphi_{V_i})$ 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, $F_i^l(\bar{a}, b) \leftarrow \mathit{body}(\varphi_{V_i})$. This implies that D satisfies $\mathit{body}(\varphi_{V_i})$ 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 $\mathcal{M}_B := (\mathcal{M}\backslash D)$ 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 $D_{\mathcal{M}} = D$ we have that every minimal legal instance has a stable model of $\Pi(\mathcal{G})$ associated. □

Proof of Theorem 2. 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 $h(S_i^l) \subseteq \bigcup_{S\in(Sec\backslash\{S_i^l\})} L(S)$, 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 $E \nsubseteq D_{\mathcal{M}}$. From Theorem 1 we have 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 $\mathcal{C} \notin \mathcal{M}'$.

From the proof of Theorem 1 we have 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, we have that 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 the ones assigned in $\mathcal{M}$ and the rest values should be the same (since $D_{\mathcal{M}'} \nsubseteq D_{\mathcal{M}}$). 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, we have that 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 $x_k \in \mathit{Const}(S_i^l)$ then there is a different section S_j^m such that $P(\ldots,x_k,\ldots) \in S_j^m$ and $x_k \in \mathit{Const}(S_j^m)$ and $x_k = a_k$.
(2) If $x_k \in \mathit{UVar}(S_i^l)$ then there are two options:
   (a) There is other section S_j^m such that $P(\ldots,x_k,\ldots) \in S_j^m$, $x_k \in \mathit{Const}(S_j^m)$ and $x_k = a_k$.
   (b) There is other section S_j^m such that $P(\ldots,x_k,\ldots) \in S_j^m$, $x_k \in \mathit{UVar}(S_j^m)$ and $(\ldots,a_k,\ldots) \in v_j$.
(3) If $x_k \in \mathit{EVar}(S_i^l)$ 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) we have that $h(x_k) = a_k$. By construction this mapping is such that $h(S_i^l) \subseteq \bigcup_{S \in (Sec \setminus \{S_i^l\})} L(S)$. We have reached a contradiction since we assumed the mapping h did not exists. Therefore we have proved Theorem 2. □

The following intermediate results refer to the refined program $\Pi(\mathcal{G})$ introduced in Section 3.2.

Lemma 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 Definition 12 of $\Pi(\mathcal{G})$. Assume that $D_{\mathcal{M}}$ is not legal. Then there must be a view V_i, with definition $\varphi_i : V_i(\bar{x}) \leftarrow \bigwedge_{u=1}^n P_u(\bar{x}_u, \bar{z}_u)$, for which $v_i \nsubseteq \varphi_i(D_{\mathcal{M}})$. More specifically, there is $\bar{a}$ such that $\bar{a} \in (v_i \setminus \varphi_i(D_{\mathcal{M}}))$. 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 (3a) 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), $P_l(\bar{a}_u, \mathbf{t_d}) \in \mathcal{M}$ 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 (3b). For a certain S_{ij}, in order to satisfy the second rule of (3b), we have to analyze two cases:

(1) $V_i(\bar{a}) \in \mathcal{M}$ and $aux_{v_{ij}}(\bar{a}') \notin \mathcal{M}$. Then, $add_{v_{ij}}(\bar{a}') \in \mathcal{M}$. From the third rule of (3b) we have 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, from the 2nd rule and since $add_{v_{ij}}(\bar{a}') \in \mathcal{M}$ we have 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 (3b). For $P_k \in S_{ij}$, there are two cases to analyze with respect to the first rule:
  (a) $\{Z_l \mid Z_l \in (\bar{X}_k \setminus \bar{X}')\} \subseteq \{Z_l \mid l \in \mathcal{L}\}$. 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 $P_k(\bar{a}_k, \mathbf{t_d}) \in \mathcal{M}$ and therefore $P_k(\bar{a}_k) \in D_{\mathcal{M}}$.
  (b) $\{Z_l \mid Z_l \in (\bar{X}_k \setminus \bar{X}')\} \nsubseteq \{Z_l \mid l \in \mathcal{L}\}$. For every $Z_{l'} \in \{\{Z_l \mid Z_l \in (\bar{X}_k \setminus \bar{X}')\} \setminus \{Z_l \mid l \in \mathcal{L}\}\}$ we have that since $l' \notin \mathcal{L}$, $var_{v_{ij} Z_{l'}}(\bar{a}_{Z_{l'}}) \in \mathcal{M}$. 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, we have that the body of the fourth rule of (3b) 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 because of the third rule of (3b) we have 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 $P_k(\bar{a}_k, \mathbf{t_d}) \in \mathcal{M}$. Then $P_k(\bar{a}_k) \in D_{\mathcal{M}}$ sharing the same existential variable that the ones generated by the previews case considered.

Then we have that $\bar{a}_{S_{ij}} \in \varphi_{i\,S_{ij}}(D_{\mathcal{M}})$.[7]

(2) $V_i(\bar{a}) \in \mathcal{M}$ and $aux_{v_{ij}}(\bar{a}') \in \mathcal{M}$. Then, $add_{v_{ij}}(\bar{a}') \notin \mathcal{M}$. From the 3rd rule of (3b) $var_{v_{ij}Z_l}(\bar{a}_{Z_l}) \in \mathcal{M}$ for all Z_l. Then, from the fourth rule of (3b) $P_k(\bar{a}_k, \mathbf{nv_{ij}}) \in \mathcal{M}$ 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 $P_k(\bar{a}_k, \mathbf{t_d}) \in \mathcal{M}$. Then $P_k(\bar{a}_k) \in D_{\mathcal{M}}$. Then we have that $\bar{a}_{S_{ij}} \in \varphi_{i\,S_{ij}}(D_{\mathcal{M}})$.

Now, since the different S_{ij} do not share existential variables we have that $\varphi_i(D_{\mathcal{M}}) = \bowtie_{S_{ij} \in V_i} \varphi_{i\,S_{ij}}(D_{\mathcal{M}})$. Then since $\bar{a}_{S_{ij}} \in \varphi_{i\,S_{ij}}(D_{\mathcal{M}})$, $\bar{a} \in \varphi_i(D_{\mathcal{M}})$. We have reached a contradiction and the lemma is proven. □

Lemma 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 as in the Definition 12 of $\Pi(\mathcal{G})$. We put the following facts into $\mathcal{M}$:

(1) $P_k(\bar{a}, \mathbf{t_d})$ for every global atom $P_k(\bar{a}) \in D$. No other atom annotated with $\mathbf{t_d}$ 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.
We have that for each view section S_i^l with an existential variable z_l,[8] such that $P_k \in S_i^l$, define the following views:

$$P_k\left(\bar{X}_k', S_i^l\right) \leftarrow \bigwedge_{P_j(\bar{X}_j) \in S_i^l} P_j(\bar{X}_j) \wedge V_i(\bar{X}),$$

[7] $\bar{a}_{S_{ij}}$ corresponds to the atom $\bar{a}$ restricted to the variables of the view φ_i that belong to S_{ij}, and $\varphi_{i\,S_{ij}}$ is the view definition φ_i restricted to the predicates in S_{ij} and its variables.

[8] The S_i^l are the view sections introduced in Section 3.1.

where S_i^l is considered as an annotation constant in the second argument of head of the view. This view will contain the information of which.

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}[9] in $\mathcal{G}$ such that $S_{ij} \supseteq S_i^m$. Then, for each section $S_i^l \in S^{P_k}$ that does not have an admissible mapping[10] such that $h(S_i^l) \subseteq \bigcup_{S \in (Sec \setminus \{S_i^l\})} L(S)$ 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), (3a), (5) and the first rule of (3b) 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 (3b). 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 (3b). This rule is satisfied by $\mathcal{M}$ because of the facts added to $\mathcal{M}$ in item (5). For the section S_i^l such have no admissible mapping such that $h(S_i^l) \subseteq \bigcup_{S \in (Sec \setminus \{S_i^l\})} L(S)$, we have that no other views can generate the facts for this section and therefore that the body of the fourth rules in (3b) 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 (3b) 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 rules in (3b) 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. □

Lemma 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 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 $D \nsubseteq D_{\mathcal{M}}$. By Lemma 2 we have that there is a model $\mathcal{M}'$ such that $D_{\mathcal{M}'} = D$. Then, $D_{\mathcal{M}'} \nsubseteq D_{\mathcal{M}}$. In particular, we have that there is an atom of a global relation, say $P_k(\bar{a}, \mathbf{t_d})$, such that $P_k(\bar{a}, \mathbf{t_d}) \in \mathcal{M}$ and $P_k(\bar{a}, \mathbf{t_d}) \notin \mathcal{M}'$. If $P_k(\bar{a}, \mathbf{t_d}) \in \mathcal{M}$ we have two options:

9 Here the S_{ij} are those appearing in Definition 12.

10 As defined in Section 3.1.

(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 $P_k(\bar{a}, \mathbf{t_d}) \notin \mathcal{M}'$.

(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 (3b) 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 $P_k(\bar{a}, \mathbf{t_d})$ then, in order to satisfy the openness of view v_i it must add a new predicate annotated with $\mathbf{t_d}$. But $D'_{\mathcal{M}} \nsubseteq D_{\mathcal{M}}$. 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 $\mathcal{G}$. □

Proof of Theorem 3. Directly from Lemmas 2 and 3. □

A.2. Obtaining the simple program from the refined program

Assume the hypothesis of Theorem 2 hold. We denote the view sections with S_i^l as in Section 3.1. 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. In consequence, for every model $\mathcal{M}$ of the refined version of $\Pi(\mathcal{G})$, we have that for all $\bar{a}$, $var_{v_{ij} Z_l}(\bar{\bar{a}}) \notin \mathcal{M}$. 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

   $$P_k(\bar{X}_k, t_o) \leftarrow V_i(\bar{X}).$$

   (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,

   $$P_k(\bar{X}_k, v_{ij}) \leftarrow add_{v_{ij}}(\bar{X}'), \quad \bigwedge_{Z_l \in (\bar{X}_k \setminus \bar{X}')} F_i^l(\bar{X}', Z_l), \quad \text{for } P_k \in S_{ij}.$$

   $$add_{v_{ij}}(\bar{X}') \leftarrow V_i(\bar{X}), \quad \text{where } \bar{X}' = \bar{X} \cap \left\{ \bigcup_{P_k \in S_{ij}} X_k \right\}.$$

4. For every predicate $F_i^l(\bar{X}', Z_l)$ introduced in (3b), the rules,

$$F_i^l(\bar{X}', Z_l) \leftarrow add_{v_{ij}Z_l}(\bar{X}'), dom(Z_l), choice((\bar{X}'), (Z_l)).$$
$$add_{v_{ij}Z_l}(\bar{X}') \leftarrow add_{v_{ij}}(\bar{X}'), \quad \text{for } l = 1, \ldots, m.$$

5. For every global relation $P(\bar{X})$ the rules

$$\begin{array}{ll} P(\bar{X}, nv_{ij}) \leftarrow P(\bar{X}, v_{hk}), & \text{for } \{(ij, hk) \mid P(\bar{X}) \in S_{ij} \text{ and } S_{hk}\}. \\ P(\bar{X}, nv_{ij}) \leftarrow P(\bar{X}, t_o), & \text{for } \{(ij) \mid P(\bar{X}) \in S_{ij}\}. \\ P(\bar{X}, t_d) \leftarrow P(\bar{X}, v_{ij}), & \text{for } \{(ij) \mid P(\bar{X}) \in S_{ij}\}. \\ P(\bar{X}, t_d) \leftarrow P(\bar{X}, t_o). & \end{array}$$

This is a positive program with choice. Because of the second rule in (3b) 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 $P(\bar{X}, t_o)$ and $P(\bar{X}, v_{ij})$ by $P(\bar{X}, t_d)$. 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

   $$P_k(\bar{X}_k, t_d) \leftarrow V_i(\bar{X}).$$

   (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,

   $$P_k(\bar{X}_k, t_d) \leftarrow V_i(\bar{X}), \quad \bigwedge_{Z_l \in (\bar{X}_k \setminus \bar{X}')} F_i^l(\bar{X}', Z_l), \quad \text{for } P_k \in S_{ij}.$$

4. For every predicate $F_i^l(\bar{X}', Z_l)$ introduced in (3b), the rules,

$$F_i^l(\bar{X}', Z_l) \leftarrow V_i(\bar{X}), \quad dom(Z_l), \quad choice\big((\bar{X}'), (Z_l)\big).$$

By merging rules (3a) and (3b), the revised version of $\Pi(\mathcal{G})$ is eventually syntactically transformed to the simple version of the program.

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.

[2] A. Abiteboul, O. Duschka, Complexity of answering queries using materialized views, in: Proc. ACM Symposium on Principles of Database Systems (PODS 98), 1998, pp. 254–263.

[3] M. Arenas, L. Bertossi, J. Chomicki, Consistent query answers in inconsistent databases, in: Proc. ACM Symposium on Principles of Database Systems (PODS 99), 1999, pp. 68–79.

[4] M. Arenas, L. Bertossi, J. Chomicki, Answer sets for consistent query answers, Theory Practice Logic Programm. 3 (4–5) (2003) 393–424.

[5] P. Barcelo, L. Bertossi, Logic programs for querying inconsistent databases, in: Proc. 5th International Symposium on Practical Aspects of Declarative Languages (PADL 03), in: Lecture Notes in Computer Science, vol. 2562, Springer, 2003, pp. 208–222.

[6] P. Barcelo, L. Bertossi, L. Bravo, Characterizing and computing semantically correct answers from databases with annotated logic and answer sets, in: Semantics of Databases, in: Lecture Notes in Computer Science, vol. 2582, Springer, 2003, pp. 1–27.

[7] R. Ben-Eliyahu, R. Dechter, Propositional semantics for disjunctive logic programs, Ann. Math. Artificial Intelligence 12 (1994) 53–87.

[8] L. Bertossi, L. Bravo, Consistent query answers in virtual data integration systems, in: Inconsistency Tolerance in Knowledge-bases, Databases and Software Specifications, Springer, submitted for publication.

[9] L. Bertossi, J. Chomicki, A. Cortes, C. Gutierrez, Consistent answers from integrated data sources, in: Proc. Flexible Query Answering Systems (FQAS 02), in: Lecture Notes in Artificial Intelligence, vol. 2522, Springer, 2002, pp. 71–85.

[10] L. Bertossi, J. Chomicki, Query answering in inconsistent databases, in: J. Chomicki, G. Saake, R. van der Meyden (Eds.), Logics for Emerging Applications of Databases, Springer, 2003.

[11] L. Bertossi, C. Schwind, Database repairs and analytic tableaux, Ann. Math. Artificial Intelligence 40 (1–2) (2004) 5–35.

[12] L. Bertossi, L. Bravo, 2004, in preparation.

[13] L. Bravo, L. Bertossi, Logic programs for consistently querying data sources, in: Proc. of 18th International Joint Conference on Artificial Intelligence (IJCAI 03), Morgan Kaufmann, 2003, pp. 10–15.

[14] A. Cali, D. Calvanese, G. De Giacomo, M. Lenzerini, Data integration under integrity constraints, in: Proc. Conference on Advanced Information Systems Engineering (CAiSE 02), in: Lecture Notes in Computer Science, vol. 2348, Springer, 2002, pp. 262–279.

[15] A. Cali, D. Lembo, R. Rosati, On the decidability and complexity of query answering over inconsistent and incomplete databases, in: Proc. ACM Symposium on Principles of Database Systems (PODS 03), 2003, pp. 260–271.

[16] A. Cali, D. Lembo, R. Rosati, Query rewriting and answering under constraints in data integration systems, in: Proc. of 18th International Joint Conference on Artificial Intelligence (IJCAI 03), Morgan Kaufmann, 2003, pp. 16–21.

[17] A. Celle, L. Bertossi, Querying inconsistent databases: algorithms and implementation, in: 'Computational Logic—CL 2000'. Stream: 6th International Conference on Rules and Objects in Databases (DOOD 00), in: Lecture Notes in Artificial Intelligence, vol. 1861, Springer, 2000, pp. 942–956.

[18] J. Chomicki, J. Marcinkowski, Minimal-change integrity maintenance using tuple deletions, arXiv.org paper cs.DB/0212004. Inform. and Comput., submitted for publication.

[19] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3) (2001) 374–425.

[20] O. Duschka, Query planning and optimization in information integration, PhD Thesis, Stanford University, December 1997.

[21] O. Duschka, M. Genesereth, A. Levy, Recursive query plans for data integration, J. Logic Programm. 43 (1) (2000) 49–73.

[22] T. Eiter, W. Faber, N. Leone, G. Pfeifer, Declarative problem-solving in DLV, in: J. Minker (Ed.), Logic-Based Artificial Intelligence, Kluwer, 2000, pp. 79–103.

[23] T. Eiter, M. Fink, G. Greco, D. Lembo, Efficient evaluation of logic programs for querying data integration systems, in: Proc. 19th International Conference on Logic Programming (ICLP 03), in: Lecture Notes in Computer Science, vol. 2916, Springer, 2003, pp. 163–177.

[24] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Comput. 9 (1991) 365–385.

[25] F. Giannotti, S. Greco, D. Sacca, C. Zaniolo, Programming with non-determinism in deductive databases, Ann. Math. Artificial Intelligence 19 (1–2) (1997) 97–125.

[26] F. Giannotti, D. Pedreschi, D. Sacca, C. Zaniolo, Non-determinism in deductive databases, in: Proc. International Conference on Rules and Objects in Databases (DOOD 91), in: Lecture Notes in Computer Science, vol. 566, Springer, 1991, pp. 129–146.

[27] F. Giannotti, D. Pedreschi, C. Zaniolo, Semantics and expressive power of nondeterministic constructs in deductive databases, J. Comput. System Sci. 62 (1) (2001) 15–42.

[28] G. Grahne, A. Mendelzon, Tableau techniques for querying information sources through global schemas, in: Proc. International Conference on Database Theory (ICDT 99), in: Lecture Notes in Computer Science, vol. 1540, Springer, 1999, pp. 332–347.

[29] J. Grant, M. Minker, A logic-based approach to data integration, Theory Practice Logic Programm. 2 (3) (2002) 323–368.

[30] G. Greco, S. Greco, E. Zumpano, A logic programming approach to the integration, repairing and querying of inconsistent databases, in: Proc. International Conference on Logic Programming (ICLP 01), in: Lecture Notes in Computer Science, vol. 2237, Springer, 2001, pp. 348–364.

[31] J. Gryz, Query rewriting using views in the presence of functional and inclusion dependencies, Inform. Syst. 24 (7) (1999) 597–612.

[32] D. Lembo, M. Lenzerini, R. Rosati, Source inconsistency and incompleteness in data integration, in: Proc. Workshop on Knowledge Representation Meets Databases (KRDB 02), 2002.

[33] M. Lenzerini, Data integration: a theoretical perspective, in: Proc. ACM Symposium on Principles of Database Systems (PODS 02), 2002, pp. 233–246.

[34] N. Leone, P. Rullo, F. Scarcello, Disjunctive stable models: unfounded sets, fixpoint semantics, and computation, Inform. and Comput. 135 (2) (1997) 69–112.

[35] N. Leone, et al., The DLV system for knowledge representation and reasoning, arXiv.org paper cs.LO/0211004, ACM Trans. Comput. Logic, submitted for publication.

[36] A. Levy, Logic-based techniques in data integration, in: J. Minker (Ed.), Logic Based Artificial Intelligence, Kluwer, 2000, pp. 575–595.

[37] V. Lifschitz, H. Turner, Splitting a logic program, in: Proc. International Conference on Logic Programming (ICLP 94), MIT Press, 1994, pp. 23–37.

[38] I. Niemela, Implementing circumscription using a tableau method, in: Proc. European Conference on Artificial Intelligence (ECAI 96), 1996, pp. 80–84.

[39] T. Przymusinski, Stable semantics for disjunctive programs, New Generation Comput. 9 (3/4) (1991) 401–424.

[40] R. Reiter, Towards a logical reconstruction of relational database theory, in: M.L. Brodie, J. Mylopoulos, J.W. Schmidt (Eds.), On Conceptual Modeling, Springer-Verlag, 1984, pp. 191–233.

[41] H. Wang, C. Zaniolo, Nonmonotonic reasoning in $\mathcal{LDL}^{++}$, in: J. Minker (Ed.), Logic-Based Artificial Intelligence, Kluwer, 2000, pp. 523–544.