Disjunctive Deductive Databases for Computing Certain and Consistent Answers to Queries from Mediated Data Integration Systems

Loreto Bravo[1] and **Leopoldo Bertossi**[2]

[1] Pontificia Universidad Catolica de Chile, Departamento de Ciencia de Computación, Santiago, Chile. lbravo@ing.puc.cl
[2] Carleton University, School of Computer Science, Ottawa, Canada. bertossi@scs.carleton.ca

Abstract. We address the problem of retrieving certain and consistent answers to queries posed to a mediated data integration system with open sources under the local-as-view paradigm using conjunctive and disjunctive view definitions. For obtaining certain answers a query program is run on top of a normal deductive database with choice that defines the class of minimal legal instances of the integration system under the cautious stable model semantics. This methodology works for all monotone Datalog queries. To compute answers to queries that are consistent wrt 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 consistent answers to any Datalog$^{\neg}$ query, for any set of universal and acyclic referential integrity constraints.

1 Introduction

When independent data sources are integrated, inconsistencies wrt to global integrity constraints (ICs) are likely to occur, specially when the sources are virtually integrated by means of a mediator, because the data sources are kept completely independent. The mediator provides a global schema as an interface, and is 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. Query answering is harder in LAV [2] (see [30] for a survey on the these and mixed approaches, and their complexity properties). On the other side, the LAV approach allows more flexibility when new sources are integrated into an existing system. However, the flexibility to add or delete sources, without having to consider the other sources in the system, makes inconsistencies wrt global ICs even more likely.

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

Inconsistency is not an exclusivity of integration systems; for several reasons a single database may become inconsistent wrt certain ICs. Restoring consistency may be undesirable, difficult or impossible [9]. 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. true in all repaired versions of the original instance (the *repairs*). In that paper and others [15, 27, 4, 5], 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. Repairs provide an auxiliary concept that allows defining the right semantics for consistent query answers; and in some of the query evaluation methodologies, also an auxiliary computational intermediate step that, for reasons of performance, has to be reduced 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 wrt the FD $R\colon X \rightarrow Y$. □

Several algorithms for deriving query plans to obtain query answers from virtual data integration systems have been proposed in the last few years (see [33] 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 [28, 19, 26]; 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 [29, 8, 11, 14].

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 to a global consistent material state. This enhanced query plan generator should produce query plans that are guaranteed to retrieve all and only consistent answers to global queries.

In [8], in this spirit and under the LAV approach, a methodology for generating query plans to compute answers to limited forms of queries that are consistent wrt 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 can be handled, namely

queries that are conjunctions of tables and universal ICs . Once the query is transformed, query plans are generated for the new query. However, [8] 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 acyclic 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* [23] and stable model semantics [22], the class of all minimal legal global instances. This database is inspired by the inverse-rules algorithm [19], 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, and then be run under the skeptical stable model semantics. It turns out that, *minimal answers*, i.e. 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 [8]. Consistent answers from an inconsistent integration systems are those that can be obtained from all the repairs of all the minimal legal instances wrt the global ICs [3, 8]. In consequence, in order to retrieve consistent answers, the specification deductive database has to be combined with another, now disjunctive deductive database that specifies the repairs of the minimal global instances. For this, we can use the deductive databases with annotations and stable model semantics that specify the repairs of single relational databases [3], as presented in [6, 5]. We have experimented with this query answering mechanism (and the computation of minimal instances and their repairs) with the *DLV* [20, 32] system, which implements the stable model semantics of disjunctive deductive databases, logic programs, and answer sets programs.

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 [11] 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 $\{R_1, R_2, ..., R_m\}$ 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 predicates and built-in predicates, like (in)equality. In particular, we will extend the global schema with a *local schema* $\mathcal{S}$, i.e. a finite set of new view predicates $V_1, V_2, ..., V_n$, that will be used to describe the relations in the local sources.

A *view*, denoted by a new predicate V, is defined by means of conjunctive query [1], i.e. an $\mathcal{L}(\mathcal{R} \cup \mathcal{S})$-formula φ_V of the form $V(\bar{t}) \leftarrow \mathit{body}(\varphi_V)$, where $\bar{t}$ is a tuple containing variables and/or constants, and $\mathit{body}(\varphi_V)$ is a conjunction of $\mathcal{R}$-atoms.

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 v (corresponding to the contents of a data source), it is possible that v is incomplete and stores only some of the tuples in $\varphi_V(D)$; and we say the view extension v is *open* wrt D [2]. Most mechanisms for deriving query plans assume that sources are open, e.g. [19].

A *source* S is a pair $\langle \varphi, v \rangle$, where φ is the view definition, and v is an extension for the view defined by φ. An *open global system* $\mathcal{G}$ is a finite set of open sources. The global schema $\mathcal{R}$ consists of the relation names that do not have a definition in the global system. The underlying domain $\mathcal{U}$ for $\mathcal{R}$ is a 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. A global system $\mathcal{G}$ defines a set of legal global instances [30].

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 query $Q(\bar{X})$ to the system, a ground tuple $\bar{t}$ is a *certain answer* to Q in $\mathcal{G}$ if for every global instance $D \in \ Linst(\mathcal{G})$, it holds $D \models Q[\bar{t}]$.[3] We denote with $Certain_{\mathcal{G}}(Q)$ the set of certain answers to Q in $\mathcal{G}$. □

The inverse-rules algorithm [19] 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 deinition, we need as many Skolem functions as existential variables in it. For example, if instead of $V_1(X, Z) \leftarrow P(X, Y)$, $R(Y, Z)$ we had, say $V_1(X, Z) \leftarrow P(X, Y)$, $R(Y, Z, W)$, we would need two Skolem functions for that view, and the inverse rules arising from that view would be $P(X, f(X, Z)) \leftarrow V_1(X, Z)$ and $R(f(X, Z), Z, g(X, Z)) \leftarrow V_1(X, Z)$. □

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

[3] $D \models Q[\bar{t}]$ 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.

2.2 Global systems and consistency

We assume that we have a set of global integrity constraints IC that is consistent as a set of logical sentences, and *generic*, in the sense that it does not entail any ground database literal. ICs used in database praxis are always generic.

Definition 3. [8] (a) Given a global system, $\mathcal{G}$, an instance D is *minimal* if $D \in \mathit{Linst}(\mathcal{G})$ and is minimal wrt set inclusion, i.e. there is no other instance in $\mathit{Linst}(\mathcal{G})$ that is a proper subset of D (as a set of atoms). We denote by $\mathit{Mininst}(\mathcal{G})$ the set of minimal legal global instances of $\mathcal{G}$ wrt set inclusion. (b) A global system $\mathcal{G}$ is *consistent* wrt IC, if for all $D \in \mathit{Mininst}(\mathcal{G})$, $D \models IC$. □

Example 5. (example 4 continued) Assume that $\mathcal{G}_2$ has the source contents $v_1 = \{V_1(a, b)\}$, $v_2 = \{V_2(a, c)\}$, resp. and that $\mathcal{U} = \{a, b, c, u, ...\}$. Then, the elements of $\mathit{Mininst}(\mathcal{G}_2)$ are of the form $D_z = \{P(a, z), R(z, b), P(a, c)\}$ for some $z \in \mathcal{U}$. The global FD $P(X, Y)\colon X \rightarrow Y$ is violated exactly in those minimal legal instances D_z for which $z \neq c$. Thus, $\mathcal{G}_2$ is inconsistent. □

Definition 4. [8] The ground tuple $\bar{a}$ is a *minimal answer* to a query Q posed to $\mathcal{G}$ if for every $D \in \mathit{Mininst}(\mathcal{G})$, $\bar{a} \in Q(D)$, where $Q(D)$ is the answer set for Q in D. The set of minimal answers is denoted by $\mathit{Minimal}_{\mathcal{G}}(Q)$. □

Clearly $\mathit{Certain}_{\mathcal{G}}(Q) \subseteq \mathit{Minimal}_{\mathcal{G}}(Q)$. For monotone queries [1], the two notions coincide [8]. Nevertheless, in Example 5 the query $\mathit{Ans}(X, Y) \leftarrow \neg P(X, Y)$ 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 wrt minimal global instances, the notion of minimal answer is particularly relevant.

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

Definition 5. [3] (a) 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 $\Delta(D, D') = (\Sigma(D) \setminus \Sigma(D')) \cup (\Sigma(D') \setminus \Sigma(D))$. (b) For database instances D, D', D'', we define $D' \leq_D D''$ if $\Delta(D, D') \subseteq \Delta(D, D'')$. □

Definition 6. (based on [3]) Let $\mathcal{G}$ be a global system and IC a set of global ICs. A *repair* of $\mathcal{G}$ wrt IC is a global database instance D', such that $D' \models IC$ and D' is $\leq_D$-minimal for some $D \in \mathit{Mininst}(\mathcal{G})$. □

Thus, a repair of a global system is a global database instance that minimally differs 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 6 we are not requiring that a repair respects the property of the sources of being open, i.e. that the extension of each view in the repair contains the corresponding view extension in the source. Thus, it may be the case that a repair – still a global instance – does not belong to $\mathit{Linst}(\mathcal{G})$. If we do not allow this this flexibility, a global system might not be repairable. Repairs are used as an auxiliary concept to define the notion of consistent answer.

Example 6. (example 1 continued) The only element in $Mininst(\mathcal{G}_1)$ is $D_0 = \{R(a,b), R(c,d), R(a,c), R(d,e)\}$, 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 $Linst(\mathcal{G}_1)$. □

Definition 7. [8] (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 wrt IC iff for every repair D of $\mathcal{G}$, $D \models Q[\bar{t}]$. (b) We denote by $Consis_{\mathcal{G}}(Q)$ the set of consistent answers to Q in $\mathcal{G}$. □

Example 7. (example 6 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 wrt IC, then $Consis_{\mathcal{G}}(Q) = Minimal_{\mathcal{G}}(Q)$. Furthermore, if the ICs are generic, then for any $\mathcal{G}$ it holds $Consis_{\mathcal{G}}(Q) \subseteq Minimal_{\mathcal{G}}(Q)$ [8]. 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 $Mininst(\mathcal{G})$ 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 [24]. 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 8. Given an open global system $\mathcal{G}$, the program, $\Pi(\mathcal{G})$, 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 view (source) predicate V_i in the system with description $V_i(\bar{X}) \leftarrow P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n)$, the rules

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

3. For every predicate $F_i^l(\bar{X}, Z_l)$ introduced in 2., the rule
$$F_i^l(\bar{X}, Z_l) \leftarrow V_i(\bar{X}), dom(Z_l), choice((\bar{X}), (Z_l)).$$ □

In this specification, the predicate $F_i^l(\bar{X}, Z_l)$ replaces the Skolem function $f_i^l(\bar{X}) = Z_l$ 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 that are not in $\bar{X}$. There is a new Skolem predicate for each pair formed by a description rule as in item 2. above and a different existentially quantified variable in it. The predicate $choice((\bar{X}), (Z_l))$ ensures that for every (tuple of) value(s) for $\bar{X}$, only one (tuple of) value(s) for Z_l is non deterministically chosen between the constants of the active domain.

Example 8. (examples 4 and 5 continued) Program $\Pi(\mathcal{G}_2)$ contains the following rules:

1. $dom(a).\ \ dom(b).\ \ dom(c).\ \ dom(u).\quad V_1(a,b).\ \ V_2(a,c).$
2. $P(X,Z) \leftarrow V_1(X,Y), F_1(X,Y,Z).$
   $R(Z,Y) \leftarrow V_1(X,Y), F_1(X,Y,Z).$
   $P(X,Y) \leftarrow V_2(X,Y).$
3. $F_1(X,Y,Z) \leftarrow V_1(X,Y), dom(Z),\ choice((X,Y),(Z)).$

In this section we will restrict ourselves to a finite domain $\mathcal{U}$, 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. □

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 [24]. The program $SV(\Pi)$ is obtained as follows:

(a) Each choice rule $r\colon\ \ H \leftarrow B, choice((\bar{X}), (Y))$ 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 9. (example 8 continued) Program $SV(\Pi(\mathcal{G}_2))$ contains the following rules:

1. $dom(a).\ \ dom(b).\ \ dom(c).\ \ dom(u).\quad V_1(a,b).\ \ V_2(a,c).$
2. $P(X,Z) \leftarrow V_1(X,Y), F_1(X,Y,Z).$
   $R(Z,Y) \leftarrow V_1(X,Y), F_1(X,Y,Z).$
   $P(X,Y) \leftarrow V_2(X,Y).$
3. $F_1(X,Y,Z) \leftarrow V_1(X,Y), dom(Z), chosen_1(X,Y,Z).$
4. $chosen_1(X,Y,Z) \leftarrow V_1(X,Y), dom(Z),\ not\ \ diffChoice_1(X,Y,Z).$
   $diffChoice_1(X,Y,Z) \leftarrow chosen_1(X,Y,Z'),\ dom(Z), Z' \neq Z.$

Its stable models are:

$\mathcal{M}_1 = \{dom(a),\ dom(b),\ dom(c),\ dom(u),\ V_1(a,b),\ V_2(a,c),\ \underline{P(a,c)},\ diffChoice_1(a,b,a),\ chosen_1(a,b,b),\ diffChoice_1(a,b,c),\ \overline{diffChoice_1(a,b,u)},\ F_1(a,b,b),\ \underline{R(b,b)},\ \underline{P(a,b)}\}$

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

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

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

The underlined elements of the models correspond to the elements in which we are interested since they are the global relations of the integration system. □

Definition 9. The global instance associated to a choice model $\mathcal{M}$ of $\Pi(\mathcal{G})$ is $D_{\mathcal{M}} = \{P(\bar{a}) \mid P \in \mathcal{R} \text{ and } P(\bar{a}) \in \mathcal{M}\}$. □

Example 10. (example 9 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

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

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

Example 11. Consider the 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 captures the minimal instances. This is important because the general program to be presented in Section 3.2 is much more complex that 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 $l = 0$, i.e. 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 $S_1^0 = \{T(X,Y)\}$.

Given a view section S_i^l, we denote by $Const(S_i^l)$, $UVar(S_i^l)$ and $EVar(S_i^l)$ the sets of constants, universal variables and existential variables, respectively, that occur in predicates in S_i^l.

Let μ, ε be two new constants, and L the following mapping from $Const(S_i^l) \cup UVar(S_i^l) \cup EVar(S_i^l$ to $Const(S_i^l) \cup \{\mu, \varepsilon\}$: (a) $h(c) = c$ for every $c \in Const(S_i^l)$; (b) $h(X) = \mu$ for every $X \in UVar(S_i^l)$; (c) $h(Z) = \varepsilon$ for every $Z \in EVar(S_i^l)$.

For a view section S_i^l, an *admissible mapping* is any mapping $h\colon Const(S_i^l) \cup UVar(S_i^l) \cup EVar(S_i^l) \rightarrow Const(S_i^l) \cup \{\mu, \varepsilon\}$, such that: (a) $h(c) = c$ for every $c \in Const(S_i^l)$; (b) $h(X) = D$ with $D \in Const(S_i^l) \cup \{\mu\}$ for every $X \in UVar(S_i^l)$; (c) $h(Z) = F$ with $F \in Const(S_i^l) \cup \{\mu, \varepsilon\}$ for every $Z \in EVar(S_i^l)$.

Notice that the mapping L above is admissible. 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. $\mathcal{S}$ denotes the set of all view sections for system $\mathcal{G}$.

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 $h(S_i^l) \subseteq \bigcup_{S \in (\mathcal{S} \setminus \{S_i^l\})} L(S)$, 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 $h(S_i^l) \subseteq \bigcup_{S \in (\mathcal{S} \setminus \{S_i^l\})} L(S)$, then it is possible to have some view contents for which the openness will be satisfied by the other sections in $\mathcal{S}$, 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 12. (example 11 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 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 13. (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 the auxiliary annotation constants that will be used as extra arguments in the database predicates. They and their intended semantics are given in the following table

annotation	atom	the tuple $P(\bar{a})$ is ...
t_d	$P(\bar{a}, t_d)$	an atom of the minimal legal instances
t_o	$P(\bar{a}, t_o)$	is an obligatory atom in all the minimal legal instances
v_i	$P(\bar{a}, v_i)$	an optional atom introduced to satisfy the openness of view v_i
nv_i	$P(\bar{a}, nv_i)$	an optional atom introduced to satisfy the openness of view that is not v_i

Definition 10. 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
   $P_k(\bar{X}_k, t_o) \leftarrow V_i(\bar{X})$.
   (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,
   $P_k(\bar{X}_k, v_{ij}) \leftarrow add_{v_{ij}}(\bar{X}'),\ \bigwedge_{Z_l \in (\bar{X}_k \setminus \bar{X}')} F_i^l(\bar{X}', Z_l)$, for $P_k \in S_{ij}$.
   $add_{v_{ij}}(\bar{X}') \leftarrow V_i(\bar{X}),\ not\ aux_{v_{ij}}(\bar{X}')$, where $\bar{X}' = \bar{X} \cap \{\bigcup_{P_k \in S_{ij}} X_k\}$.
   $aux_{v_{ij}}(\bar{X}') \leftarrow \bigwedge_{i=1}^{m} var_{v_{ij}Z_l}(\bar{X}_{Z_l})$.
   $var_{v_{ij}Z_l}(\bar{X}_{Z_l}) \leftarrow \bigwedge_{P_k \in S_{ij} \wedge Z_l \in \bar{X}_k} P_k(\bar{X}_k, nv_{ij})$, where $\bar{X}_{Z_l} = \{\bigcup_{P_k \in S_{ij} \wedge Z_l \in \bar{X}_k} X_k\}$, for $l = 1, \ldots m$.
4. For every predicate $F_i^l(\bar{X}', Z_l)$ introduced in 3.(b), the rules,
   $F_i^l(\bar{X}', Z_l) \leftarrow add_{v_{ij}Z_l}(\bar{X}'), dom(Z_l), choice((\bar{X}'), (Z_l))$.
   $add_{v_{ij}Z_l}(\bar{X}') \leftarrow add_{v_{ij}}(\bar{X}'),\ not\ aux_{v_{ij}Z_l}(\bar{X}')$, for $l = 1, \ldots m$.
   $aux_{v_{ij}Z_l}(\bar{X}') \leftarrow var_{v_{ij}Z_l}(\bar{X}_{Z_l}), \bigwedge_{Z_k \neq Z_l \wedge Z_k \in \bar{X}_{Z_l}} F_i^k(\bar{X}', Z_k)$, for $l = 1, \ldots\ m$.
5. For every global relation $P(\bar{X})$ the rules
   $P(\bar{X}, nv_{ij}) \leftarrow P(\bar{X}, v_{hk}),\quad$ for $\{(ij, hk) | P(\bar{X}) \in S_{ij} \cap S_{hk}\}$.
   $P(\bar{X}, nv_{ij}) \leftarrow P(\bar{X}, t_o),\quad$ for $\{(ij) | P(\bar{X}) \in S_{ij}\}$.
   $P(\bar{X}, t_d) \leftarrow P(\bar{X}, v_{ij}),\quad$ for $\{(ij) | P(\bar{X}) \in S_{ij}\}$.
   $P(\bar{X}, t_d) \leftarrow P(\bar{X}, t_o)$. □

Example 14. (example 11 continued) The refined program $\Pi(\mathcal{G}_3)$ is:

$$dom(a). \quad dom(c). \tag{1}$$
$$v_1(a). \quad v_2(a,c). \tag{2}$$
$$P(X,Z,v_1) \leftarrow add_{v_1}(X), F_z(X,Z). \tag{3}$$
$$add_{v_1}(X) \leftarrow v_1(X),\ not\ aux_{v_1}(X). \tag{4}$$
$$aux_{v_1}(X) \leftarrow var_{v_1 z}(X,Z). \tag{5}$$
$$var_{v_1 z}(X,Z) \leftarrow P(X,Z,nv_1). \tag{6}$$
$$F_z(X,Z) \leftarrow add_{v_1}(X), dom(Z), chosen_{v_1 z}(X,Z). \tag{7}$$
$$chosen_{v_1 z}(X,Z) \leftarrow add_{v_1}(X), dom(Z),\ not\ diffchoice_{v_1 z}(X,Z). \tag{8}$$
$$diffchoice_{v_1 z}(X,Z) \leftarrow chosen_{v_1 z}(X,Z'), dom(Z), Z' \neq Z. \tag{9}$$
$$P(X,Y,t_o) \leftarrow v_2(X,Y). \tag{10}$$
$$P(X,Y,nv_1) \leftarrow P(X,Y,t_o). \tag{11}$$
$$P(X,Y,t_d) \leftarrow P(X,Y,v_1). \tag{12}$$
$$P(X,Y,t_d) \leftarrow P(X,Y,t_o). \tag{13}$$

Rules (3) to (6) 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 (3) with a Z value given by the function predicate $F_z(\bar{a}, Z)$. This function predicate is calculated by rules (7) to (9). Rule (10) enforces the satisfaction of the openness of V_2 by adding obligatory atoms to predicate P and rule (11) stores this atoms with the annotation nv_1 implying that they were added by a view different from V_1. The last two rules gather with an annotation 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 $\{dom(a), dom(c),\ v_1(a),\ v_2(a,c),\ \underline{P(a,c,t_d)},\ P(a,c,t_o),\ P(a,c,\ nv_1),\ aux_{v_1}(a)\}$, 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 $P(\bar{a}, t_d) \in \mathcal{M}\} \in Mininst(\mathcal{G})$. 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 exactly 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 using disjunctive logic programs with stable model semantics. That approach works for arbitrary universal and acyclic referential ICs, in the sense that the repairs of the database correspond to the stable models of the program. 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 $\mathbf{t_d}$ ($\mathbf{f_d}$). 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.

Example 15. Consider the integrity constraint $\forall x(P(x) \rightarrow R(x))$, and the inconsistent database instance $r = \{P(a)\}$. 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 1. and 4. 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 2. 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 6. below, that are used to interpret the models as database repairs. The following is the program:

1. $P(x, \mathbf{f}^\star) \leftarrow P(x, \mathbf{f_a}).$ $\quad P(x, \mathbf{t}^\star) \leftarrow P(x, \mathbf{t_a}).$
   $P(x, \mathbf{t}^\star) \leftarrow P(x, \mathbf{t_d}).$ (similarly for R)
2. $P(x, \mathbf{f_a}) \vee R(x, \mathbf{t_a}) \leftarrow P(x, \mathbf{t}^\star), R(x, \mathbf{f}^\star).$
3. $P(a, \mathbf{t_d}) \leftarrow.$
4. $P(x, \mathbf{f}^\star) \leftarrow \mathit{not}\ P(x, \mathbf{t_d}).$ $\quad R(x, \mathbf{f}^\star) \leftarrow \mathit{not}\ R(x, \mathbf{t_d}).$
5. $\leftarrow P(\bar{x}, \mathbf{t_a}), P(\bar{x}, \mathbf{f_a}).$ $\quad \leftarrow R(\bar{x}, \mathbf{t_a}), R(\bar{x}, \mathbf{f_a}).$
6. $P(x, \mathbf{t}^{\star\star}) \leftarrow P(x, \mathbf{t_a}).$ $\quad P(x, \mathbf{f}^{\star\star}) \leftarrow P(x, \mathbf{f_a}).$

$P(x, \mathbf{t}^{\star\star}) \leftarrow P(x, \mathbf{t_d}),\ not\ P(x, \mathbf{f_a})$.
$P(x, \mathbf{f}^{\star\star}) \leftarrow\ not\ P(x, \mathbf{t_d}),\ not\ P(x, \mathbf{t_a})$. (similarly for R)

Only rules 2. depend on the ICs. They say how to repair them when violations are found. Rules 3. contain the database atoms. Rules 4. capture the *closed world assumption* (CWA) [36]. Rules 5. are denial program constraints to discard models that contain an atom annotated with both $\mathbf{t_a}$ and $\mathbf{f_a}$. The program has two stable models: $\{P(a, \mathbf{t_d}), P(a, \mathbf{t}^\star), R(a, \mathbf{f}^\star), R(a, \mathbf{t_a}), \underline{P(a, \mathbf{t}^{\star\star})}, R(a, \mathbf{t}^\star), \underline{R(a, \mathbf{t}^{\star\star})}\}$, and $\{P(a, \mathbf{t_d}), P(a, \mathbf{t}^\star), P(a, \mathbf{f}^\star), R(a, \mathbf{f}^\star), \underline{P(a, \mathbf{f}^{\star\star})}, \underline{R(a, \mathbf{f}^{\star\star})}, P(a, \mathbf{f_a})\}$, the first one saying (look at underlined atoms) that $R(a)$ is inserted into the database; the second one, that $P(a)$ is deleted. □

In can be proved [6] in the contenxt 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 (the latter are of the form $\forall \bar{x}(P(\bar{x}) \rightarrow \exists y Q(\bar{x}', y))$, with $\bar{x}' \subseteq \bar{x}$). This property 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. 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 [10].

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 11. The *repair program*, $\Pi(\mathcal{G}, IC)$, of $\mathcal{G}$ wrt IC contains the following clauses:

1. The same rules as in Definition 10.
2. For every predicate $P \in \mathcal{R}$, the clauses
   $P(\bar{X}, \mathbf{t}^\star) \leftarrow P(\bar{X}, \mathbf{t_d}),\ dom(\bar{X})$.[4]
   $P(\bar{X}, \mathbf{t}^\star) \leftarrow P(\bar{X}, \mathbf{t_a}),\ dom(\bar{X})$.
   $P(\bar{X}, \mathbf{f}^\star) \leftarrow P(\bar{X}, \mathbf{f_a}),\ dom(\bar{X})$.
   $P(\bar{X}, \mathbf{f}^\star) \leftarrow dom(\bar{X}),\ not\ P(\bar{X}, \mathbf{t_d})$.
3. For every first-order global universal IC of the form $\overline{\forall}(Q_1(\bar{y}_1) \vee \cdots \vee Q_n(\bar{Y}_n) \leftarrow P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m) \wedge \varphi)$, 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
   $P(\bar{X}, \mathbf{f_a}) \vee Q(\bar{X}', null, \mathbf{t_a}) \leftarrow P(\bar{X}, \mathbf{t}^\star), not\ aux(\bar{X}'),\ not\ Q(\bar{X}', null, \mathbf{t_d}), dom(\bar{X})$.
   $aux(\bar{X}') \leftarrow Q(\bar{X}', Y, \mathbf{t_d}),\ not\ Q(\bar{X}', Y, \mathbf{f_a}),\ dom(\bar{X}', Y)$.
   $aux(\bar{X}') \leftarrow Q(\bar{X}', Y, \mathbf{t_a}),\ dom(\bar{X}', Y)$.

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

5. For every predicate $P \in \mathcal{R}$, the interpretation clauses:
$P(\bar{a}, \mathbf{f}^{\star\star}) \leftarrow P(\bar{a}, \mathbf{f_a})$.
$P(\bar{a}, \mathbf{f}^{\star\star}) \leftarrow \mathit{not}\ P(\bar{a}, \mathbf{t_d}),\ \mathit{not}\ P(\bar{a}, \mathbf{t_a})$.
$P(\bar{a}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{a}, \mathbf{t_a})$.
$P(\bar{a}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{a}, \mathbf{t_d}),\ \mathit{not}\ P(\bar{a}, \mathbf{f_a})$. □

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, we consider that the new constant $\mathit{null} \notin \mathcal{U}$, in particular $\mathit{dom}(\mathit{null})$ is not a fact. Optimizations of the repair part of the program, like avoiding the materialization of the CWA, are presented 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 12. The global instance associated to a choice model $\mathcal{M}$ of $\Pi(\mathcal{G}, IC)$ is $D_{\mathcal{M}} = \{P(\bar{a}) \mid P \in \mathcal{R} \text{ and } P(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}\}$. □

The repair program can be split [34] 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 $\mathcal{G}$ wrt IC. Furthermore, the repairs obtained in this way are all the repairs of $\mathcal{G}$ wrt IC. □

5 Consistent Answers

Now, we can obtain the answers to queries posed to a system $\mathcal{G}$ that are consistent wrt to IC. 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 $\mathit{not}\ P(\bar{t})$, 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 16. (example 9 continued) Consider the global symmetry integrity constraint $sim\colon\ \forall x \forall y(R(x,y) \rightarrow R(y,x))$ on $\mathcal{G}_2$. We want the consistent answers to the query $Q\colon P(x,y)$. 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 written in *DLV* syntax:

```
%begin subprogram for minimal instances
    dom(a).     dom(b).
    dom(c).     dom(u).
    v1(a,b).    v2(a,c).
    P(X,Y,nv1) :- P(X,Y,to).
    P(X,Y,nv2) :- P(X,Y,v1).
    P(X,Y,td) :- P(X,Y,v1).
    P(X,Y,td) :- P(X,Y,to).
    R(X,Y,td) :- R(X,Y,v1).

    %Specification of V1
    P(X,Y,v1) :- addv1(X,Z), fv1y(X,Z,Y).
    R(Y,Z,v1) :- addv1(X,Z), fv1y(X,Z,Y).
    addv1(X,Z) :- v1(X,Z), not auxv1(X,Z).
    auxv1(X,Z) :- varv1y(X,Y,Z).
    varv1y(X,Y,Z) :- P(X,Y,nv1),R(Y,Z,nv1).
    fv1y(X,Z,Y) :- addv1y(X,Z), dom(Y), chosenv1(X,Z,Y).
    chosenv1(X,Z,Y) :- addv1y(X,Z), dom(Y),not diffchoicev1(X,Z,Y).
    diffchoicev1(X,Z,Y) :- chosenv1(X,Z,YY), dom(Y), YY!=Y.
    addv1y(X,Z):- addv1(X,Z), not auxY(X,Z).
    auxY(X,Z):- varv1y(X,Y,Z).

    %Specification of V2
    P(X,Y,to) :- v2(X,Y).

%begin repair subprogram
    P(X,Y,ts) :- P(X,Y,ta), dom(X), dom(Y).
    P(X,Y,ts) :- P(X,Y,td), dom(X), dom(Y).
    P(X,Y,fs) :- dom(X), dom(Y), not P(X,Y,td).
    P(X,Y,fs) :- P(X,Y,fa), dom(X), dom(Y).

    R(X,Y,ts) :- R(X,Y,ta), dom(X), dom(Y).
    R(X,Y,ts) :- R(X,Y,td), dom(X), dom(Y).
    R(X,Y,fs) :- dom(X), dom(Y), not R(X,Y,td).
    R(X,Y,fs) :- R(X,Y,fa), dom(X), dom(Y).

    R(X,Y,fa) v R(Y,X,ta) :-  R(X,Y,ts), R(Y,X,fs), dom(X), dom(Y).

    P(X,Y,tss) :- P(X,Y,ta), dom(X), dom(Y).
    P(X,Y,tss) :- P(X,Y,td), dom(X), dom(Y), not P(X,Y,fa).
```

```
P(X,Y,fss) :- P(X,Y,fa), dom(X), dom(Y).
P(X,Y,fss) :- dom(X), dom(Y), not P(X,Y,td), not P(X,Y,ta).

R(X,Y,tss) :- R(X,Y,ta), dom(X), dom(Y).
R(X,Y,tss) :- R(X,Y,td), dom(X), dom(Y), not R(X,Y,fa).
R(X,Y,fss) :- R(X,Y,fa), dom(X), dom(Y).
R(X,Y,fss) :- dom(X), dom(Y), not R(X,Y,td), not R(X,Y,ta).

:- R(X,Y,ta), R(X,Y,fa).
:- P(X,Y,ta), P(X,Y,fa).
```

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 10; (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 [37]). 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 [17, 32] 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 wrt the disjunctive normal program $\Pi(\mathcal{G}, IC)$ that specifies the repair of minimal instances is Π_2^P-complete in data complexity [17], what matches the complexity of consistent query answering [9].

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, 31]. This *coNP*-completeness result can be extended to some cases where both universal and RICs are considered. It is possible to show [10] 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. formulas of the form $\forall \bar{x}(p(\bar{x}) \rightarrow \exists y q(\bar{x}', y))$, with $\bar{x}' \subseteq \bar{x}$, but without cycles in the dependency graph.

This case includes the usual integrity constraints found in database practice, like (non cyclic) foreign key constraints. This result can be further extended [10], but there are theoretical limits. In [16, 13] some cases where functional dependencies and referential integrities coexist are presented, for which the problem of CQA becomes Π_2^P-complete.

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 8, an infinite number of facts. Our soundness and completeness theorems hold. However, in the logic programs we have presented in the examples we had a finite domain, c.f. Example 8 (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 17. 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 $Certain_{\mathcal{G}_4}(Q) = \{a\}$. 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 $\square$

In Example 8, 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 $Mininst(\mathcal{G}_2) = \{\{P(a,c),\ P(a,z),\ R(z,a)\} \mid z \in \mathcal{U} = \{a,b,c,...\}\}$.

Since we want only the certain answers, those that can be obtained from all the stable models, it is easy to see that the values taken by the "free variables", like z above, will not appear in a certain answer. However, the absence of the extra, new constants may sanction as certain some answers that are not if the domain is restricted to the active domain (see Example 17). 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 8).

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 18. Given an integration system $\mathcal{G}_5$:

$V_1(X,Y) \leftarrow P(X,Z_0), R(Z_0,Y).$ $\quad \{V_1(a,b)\}$

$V_2(X,Y) \leftarrow P(X,Z_1), R(Z_2,Y).$ $\quad \{V_2(a,b), V_2(c,d)\}$

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 wrt 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 wrt 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 [19]).

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 $F(X,Y) \leftarrow V(X)$, $dom(Y)$, $choice((X),(Y))$.

In this case, the program will have the same rules $\mathcal{V}^{-1}$ as in the inverse rules algorithm. The resulting definite program is positive and, therefore, its stable model corresponds to the minimal model. That model will have atoms with instantiated Skolem functions, and can be seen as a compact representation of the collection of stable models of the choice program, in the sense that the latter can be recovered by considering the different ways in which the Skolem functions can be defined in the underlying domain.

If a query is posed 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 19. Consider an integration system $\mathcal{G}$ with

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

The following is the program with Skolem functions in *DLV* syntax:[5]

```
%V1
    P(X,f2(X),f3(X),v1) :- addv11(X), addv1y(X),addv1z(X).
    S(Y,v1) :- addv11(X), fv1y(X,Y).
    addv11(X) :- v1(X), not auxv11(X).
    auxv11(X) :- varv1y(X,Y,Z),varv1z(X,Y,Z).
    varv1y(X,Y,Z) :- P(X,Y,Z,nv1),S(Y,nv1).
    varv1z(X,Y,Z) :- P(X,Y,Z,nv1).
    addv1y(X):- addv11(X), not auxY(X).
    auxY(X):- varv1y(X,Y,Z),fv1z(X,Z).
    addv1z(X):- addv11(X), not auxZ(X).
```

[5] Except for the function symbols, which are not supported by *DLV*.

```
    auxZ(X):- varv1z(X,Y,Z),fv1y(X,Y).
%V2
    P(X,Y,f1(X,Y),v2) :- addv21(X,Y), addv2z(X,Y).
    addv21(X,Y) :- v2(X,Y),not auxv21(X,Y).
    auxv21(X,Y) :- varv2z(X,Y,Z).
    varv2z(X,Y,Z) :- P(X,Y,Z,nv2).
    addv2z(X,Y):- addv21(X,Y), not auxv2Z(X,Y).
    auxv2Z(X,Y):- varv2z(X,Y,Z).

    P(X,Y,Z,nv1) :- P(X,Y,Z,v2).
    P(X,Y,Z,nv2) :- P(X,Y,Z,v1).
    P(X,Y,Z,td) :- P(X,Y,Z,v1).
    P(X,Y,Z,td) :- P(X,Y,Z,v2).
    S(Y,td):- S(Y,v1).
```
□

The stable models of the refined program with Skolem functions are calculated under the *unique names assumption* [36]. 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 20. (example 19 continued) The minimal instances of this integration system can be represented by $\{\{P(a, e, f_1(a, e)), P(a, f_2(a), f_3(a)), S(f_2(a))\} \mid f_1(a, e) \in \mathcal{U}, f_3(a) \in \mathcal{U}, f_2(a) \in \mathcal{U} \setminus \{e\}\} \bigcup \{\{P(a, e, f_1(a, e)), S(e)\} \mid f_1(a, e) \in \mathcal{U}\}$. 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, $f_2(a)$ 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 $P(X, f_2(X), f_3(X), v1)$. In consequence, the program will generate the stable model $\{P(a, e, f_1(a, e)), P(a, f_2(a), f_3(a)), S(f_2(a))\}$, that represents a proper superclass of the minimal legal instances. □

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 21. (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 wrt $IC: \forall x \forall y (R(X,Y) \rightarrow R(Y,X))$.

The repair program $\Pi(\mathcal{G}, IC)$ has the rule

$$R(X, Y, f_a) \vee R(Y, X, t_a) \leftarrow R(X, Y, t_s), R(Y, X, f_s),$$

that will produce the set of repairs $DB_{\mathcal{M}_1} = \{P(a, f(a,b)), P(a,c)\}$ and $DB_{\mathcal{M}_2} = \{P(a, f(a,b)), R(f(a,b), b), R(b, f(a,b)), P(a,c)\}$, 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 wrt IC. □

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 [18], 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 13. 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 $\mathit{Linst}(\mathcal{G}) = \{D \text{ instance over } \mathcal{R} \mid v_i \subseteq \bigcup_k \varphi_{ik}(D), \text{ for } i = 1, \ldots, n\}$. □

Example 22. 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 [18] 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 23. (example 22 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 23, the following rule needs to be imposed

$$(R(X,d) \wedge S(d)) \vee T(X,d) \leftarrow V(X,d), \tag{14}$$

which is equivalent to the pair of Disjunctive Datalog rules

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

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

For each set of truly disjunctive views, rules like (15) and (16) will have to be satisfied by the legal instances. These remarks motivate the following program as an specification of the minimal legal instances.

Definition 14. 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

$\mathcal{V}_{i1}: \quad V_i(\bar{X}_1) \leftarrow P_{11}(\bar{X}_{11}), \ldots, P_{1n}(\bar{X}_{1n_1})$

$\cdots$

$\mathcal{V}_{ik}: \quad V_i(\bar{X}_k) \leftarrow P_{k1}(\bar{X}_{k1}), \ldots, P_{kn}(\bar{X}_{kn_k}),$

where the variables in each view are different (fresh), for its more general witness $\bar{W}$ and its most general exclusion condition φ, the rules

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

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 24. (example 23 continued) The program $\Pi^\vee(\mathcal{G}_7)$ is:

$$dom(a).\ \ dom(b). \qquad dom(c).\ \ dom(d). \tag{17}$$
$$R(X, Y) \leftarrow V_1(X, Y), Y \neq d. \tag{18}$$
$$S(Y) \leftarrow V_1(X, Y), Y \neq d. \tag{19}$$
$$T(X, d) \vee R(X, Y) \leftarrow V_1(X, Y). \tag{20}$$
$$T(X, d) \vee S(Y) \leftarrow V_1(X, Y). \tag{21}$$
$$S(X) \leftarrow V_2(X). \tag{22}$$

Rules (18)-(19) and (20)- (21) represent, respectively, the first and second truly disjunctive set for source v_1. The last rule 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. Wrt the ICs and queries this approach can handle, the solution is sound and complete for combinations of universal ICs and non-cyclic 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 [8] 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, 31]. Transformations are shown in [6].

The program for CQA can be split [34] 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 [25]. The second layer, i.e. the repair part, is *locally stratified* [35]. 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.

For CQA from integration systems we have successfully experimented with *DLV* [20, 32]. 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 [21].

Wrt related papers, query answering in mediated integration systems *under the assumption* that certain global ICs hold has been treated in [28, 19, 26, 12]. 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, 27, 5].

In [8], 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 [29]. 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 [14] and still under the GAV approach, this work is extended by introducing rewriting techniques to retrieve the consistent query answers without constructing the "repairs".

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 our treatment of CQA in integration systems to the mixed case where open, closed and sources that are both open and closed (clopen) coexist; 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.

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.

Acknowledgments: Research funded by DIPUC, CONICYT, Carleton University Start-Up Grant 9364-01, NSERC Grant 250279-02. 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 [11], a first version of this paper, for useful remarks.

References

1. Abiteboul, S.; Hull, R. and Vianu, V. *Foundations of Databases*. Addison-Wesley, 1995.
2. Abiteboul, A. and Duschka, O. Complexity of Answering Queries Using Materialized Views. In *Proc. ACM Symposium on Principles of Database Systems (PODS 98)*, 1998, pp. 254-263.

3. Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM Symposium on Principles of Database Systems (PODS 99)*, 1999, pp. 68–79.
4. Arenas, M.; Bertossi, L.; and Chomicki, J. Answer Sets for Consistent Query Answers. Theory and Practice of Logic Programming, 2003, 3(4-5), pp. 393-424.
5. Barcelo, P.; and Bertossi, L. Logic Programs for Querying Inconsistent Databases. In *Proc. Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 03)*. Springer Lecture Notes in Computer Science 2562, 2003, pp. 208–222.
6. Barcelo, P., Bertossi, L. and Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In 'Semantics of Databases', Springer LNCS 2582, 2003, pp. 1–27.
7. Ben-Eliyahu, R. and Dechter, R. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics in Artificial Intelligence*, 1994, 12:53-87.
8. Bertossi, L., Chomicki, J., Cortes, A. and Gutierrez, C. Consistent Answers from Integrated Data Sources. In *Proc. Flexible Query Answering Systems (FQAS 02)*, Springer LNAI 2522, 2002, pp. 71–85.
9. Bertossi, L.; and Chomicki, J. Query Answering in Inconsistent Databases. In 'Logics for Emerging Applications of Databases', J. Chomicki, G. Saake and R. van der Meyden (eds.), Springer, 2003.
10. Bertossi, L. and Bravo, L. On Theoretical Aspects of Consistent Query Answering. In preparation, 2003.
11. Bravo, L. and Bertossi, L. Logic Programs for Consistently Querying Data Sources In *Proc. of the Eighteenth International Joint Conference on Artificial Intellience (IJCAI 03)*, Morgan Kaufmann, 2003, pp. 10–15.
12. Cali, A.; Calvanese, D.; De Giacomo, G. and Lenzerini, M. Data integration Under Integrity Constraints. In *Proc. Conference on Advanced Information Systems Engineering (CAiSE 02)*, Springer LNCS 2348, 2002, pp. 262–279.
13. Cali, A., Lembo, D. and Rosati, R. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proc. ACM Symposium on Principles of Database Systems (PODS 03)*, 2003, pp. 260-271.
14. Cali, A., Lembo, D., and Rosati, R. Query Rewriting and Answering under Constraints in Data Integration Systems. In *Proc. of the Eighteenth International Joint Conference on Artificial Intellience (IJCAI 03)*. Morgan Kaufmann, 2003, 99. 16-21.
15. Celle, A.; and Bertossi, L. Querying Inconsistent Databases: Algorithms and Implementation. In *'Computational Logic - CL 2000'. Stream: 6th International Conference on Rules and Objects in Databases (DOOD 00)*, Springer Lecture Notes in Artificial Intelligence 1861, 2000, pp. 942–956.
16. Chomicki, J.; and Marcinkowski, J. Minimal-Change Integrity Maintenance Using Tuple Deletions. arXiv.org paper cs.DB/0212004.
17. Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity And Expressive Power Of Logic Programming. *ACM Computer Surveys*. 2001, 33(3), 374-425.
18. Duschka, O. *Query Planning and Optimization in Information Integration*. PhD Thesis, Stanford University, December 1997.
19. Duschka, O., Genesereth, M. and Levy, A. Recursive Query Plans for Data Integration. *Journal of Logic Programming*, 2000, 43(1):49-73.
20. Eiter, T., Faber, W.; Leone, N. and Pfeifer, G. Declarative Problem-Solving in DLV. In 'Logic-Based Artificial Intelligence', J. Minker (ed.), Kluwer, 2000, pp. 79-103.
21. Eiter, T., Fink, M., Greco, G. and Lembo, D. Efficient Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In *Proc. 19th International Conference on Logic Programming (ICLP 03)*, Springer LNCS, 2003.

22. Gelfond, M. and Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365–385.
23. Giannotti, F., Greco, S., Sacca, D. and Zaniolo, C. Programming with Nondeterminism in Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 1997, 19(1-2):97–125.
24. Giannotti, F.; Pedreschi, D.; Sacca, D. and Zaniolo, C. Non-Determinism in Deductive Databases. In *Proc. International Conference on Rules and Objects in Databases (DOOD 91)*, Springer LNCS 566, 1991, pp. 129–146.
25. Giannotti, F.; Pedreschi, D. and Zaniolo, C. Semantics and Expressive Power of Nondeterministic Constructs in Deductive Databases. *J. Computer and System Sciences*, 2001, 62(1):15–42.
26. Grant, J. and Minker, M. A Logic-based Approach to Data Integration. *Theory and Practice of Logic Programming*, 2002, 2(3):323-368.
27. Greco, G., Greco, S. and Zumpano, E. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *Proc. International Conference on Logic Programming (ICLP 01)*, Springer LNCS 2237, 2001, pp. 348–364.
28. Gryz, J. Query Rewriting Using Views in the Presence of Functional and Inclusion Dependencies. *Information Systems*, 1999, 24(7):597–612.
29. Lembo, D.; Lenzerini, M. and Rosati, R. Source Inconsistency and Incompleteness in Data Integration. In *Proc. Workshop on Knowledge Representation meets Databases (KRDB 02)*, 2002.
30. Lenzerini, M. Data Integration: A Theoretical Perspective. In *Proc. ACM Symposium on Principles of Database Systems (PODS 02)*, 2002, pp. 233-246.
31. Leone, N., Rullo, P. and Scarcello, F. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation*, 1997, 135(2):69-112.
32. Leone, N. et al. The DLV System for Konwledge Representation and Reasoning. arXiv.org paper cs.LO/0211004. To appear in *ACM Transactions on Computational Logic*.
33. Levy, A. Logic-Based Techniques in Data Integration. Chapter in 'Logic Based Artificial Intelligence', J. Minker (ed.), Kluwer, 2000, pp. 575-595.
34. Lifschitz, V. and Turner, H. Splitting a Logic Program. In *Proc. International Conference on Logic Programming (ICLP 94)*. The MIT Press, 1994, pp. 23–37.
35. Przymusinski, T. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 1991, 9(3/4):401–424.
36. Reiter, R. Towards a Logical Reconstruction of Relational Database Theory. In 'On Conceptual Modeling', M.L. Brodie, J. Mylopoulos, and J.W. Schmidt (eds.). Springer-Verlag, 1984, pp. 191-233.
37. Wang, H. and Zaniolo, C. Nonmonotonic Reasoning in $\mathcal{LDL}^{++}$. In 'Logic-Based Artificial Intelligence', J. Minker (ed.), Kluwer 2000, pp. 523-544.

A Appendix

A.1 Proof of Results

Proof of Theorem 1: The program $SV(\Pi(\mathcal{G}))$ can be split [34] into the bottom program Π_B, that contains the facts and rules in 1. and 3. in Definition 8, 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 program with the rules in (2) that borrows the facts for its extensional, i.e. non defined predicates from $\mathcal{M}_B$), 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 [37]. Since Π_T is a positive program (without the choice operator), the stable models of the programs $\Pi_T^{\mathcal{M}_B}$ are their the minimal models.

We need to prove that the class of instances associated to models of the form $\mathcal{M}_B \cup \mathcal{M}_T^B$ contains only legal instances and is a superclass of the class of minimal instances. First we prove:

$$\{D_{\mathcal{M}} \mid \mathcal{M} \text{ is a choice model of } \Pi(\mathcal{G})\} \subseteq \mathit{Linst}(\mathcal{G}). \tag{23}$$

Assume that there is a stable model $\mathcal{M}$ of $\Pi(\mathcal{G})$ such that its associated database $D_{\mathcal{M}}$ is not a legal instance. Then there is a view V_i for which $v_i \not\subseteq \varphi_i(D_{\mathcal{M}})$, that is, for some $\bar{a}$:

- $\bar{a} \in v_i$, and then by 1. in Definition 8, $V_i(\bar{a})$ 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 $\neg\exists\bar{z}(P_1(\bar{a}'_1, \bar{z}_1) \wedge \ldots \wedge P_n(\bar{a}'_n, \bar{z}_n))$, for $\bar{a}_i \subseteq \bar{a}$, and $\bar{z}_i \subseteq \bar{z}$. This is equivalent to

$$\forall\bar{z} \;\; (\neg P_1(\bar{a}'_1, \bar{z}_1) \vee \ldots \vee \neg P_n(\bar{a}'_n, \bar{z}_n)) \tag{24}$$

  being true in $\mathcal{M}$.

Since the stable models containing the extensions for the P_j are minimal models (and then the P_js can get their tuples from the bodies of the rules defining them alone), from (24) we have that in $\mathcal{M}$ it holds

$$\forall\bar{z} \;\; (\neg V_i(\bar{a}) \vee \bigvee_{il} \neg F_i^l(\bar{a}, z_l)). \tag{25}$$

(There may be several rules defining each predicate P_j, that is why may have here Skolem functions coming from other view definitions.)

Since $V_i(\bar{a}) \in \mathcal{M}$, we can apply the rules 3. in Definition 8, and we get atoms $F_i^l(\bar{a}, b) \in \mathcal{M}$ for some b's in the domain. In consequence, (25) is false in $\mathcal{M}$. We have reached a contradiction; and (23) is established.

Now we want to prove:

$$\mathit{Mininst}(\mathcal{G}) \subseteq \{D_{\mathcal{M}} \mid \mathcal{M} \text{ is a choice model of } \Pi(\mathcal{G})\}. \tag{26}$$

We will now prove that every minimal legal D 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. It contains extensions for the global relations only. Let us define a Herbrand structure for the program $\Pi(\mathcal{G})$ containing the following ground atoms:

1. The atoms in D;
2. $V_i(\bar{a})$ whenever v_i is a source extension in $\mathcal{G}$ and $\bar{a} \in v_i$;
3. $dom(a)$ for every constant $a \in \mathcal{U}$;
4. For each view V_i, consider the rules $F_i^l(\bar{x}, z_l) \leftarrow body(\varphi_{V_i}),\ l = 1, \ldots$, evaluate the bodies according to the atoms in 1. When the body is true, add to $\mathcal{M}$ the corresponding atom in the head. Due to the minimality of D, the predicates become functional.
5. If for a view V_i, $\bar{a} \in v_i$, then put $choice(\bar{a}, b)$ into $\mathcal{M}$ when $F_i^l(\bar{a}, b)$ belongs to $\mathcal{M}$.

Now we have to prove that $\mathcal{M}$ is a stable model of $\Pi(\mathcal{G})$. This can be shown by proving, first, that $\mathcal{M}_B := (\mathcal{M} \setminus D)$ is a stable model of Π_B, and, next, that the set containing the atoms in 1., 2. and 4. above is a minimal model of $\Pi_T^{\mathcal{M}_B}$.

$\mathcal{M}_B$ will contain, by construction, all the facts of Π_B. Since D is legal, for every view V_i, it holds $v_i \subseteq \varphi(D)$.

The minimality of D implies that for every $V_i(\bar{a}) \in \mathcal{M}_B$, the body of the rule defining V_i becomes true in D; then there is some value b_i^l for which $F_i^l(\bar{a}, b_i^l)$ is an atom in $\mathcal{M}_B$ (see item 4. above). Since the rules 3. of Π_B are defined using the choice operator, there will be one choice model that produces the $F_i^l(\bar{a}, b)$ of $\mathcal{M}$.

Since the choice predicate chooses exactly one value for each element in the view, and $\mathcal{M}_B$ is a minimal model of the positive program obtained from Π_B by adding as facts the atoms in the extension of the choice predicate in $\mathcal{M}$, we have that $\mathcal{M}_B$ is a choice model of Π_B.

Now we have to verify that $\mathcal{M}_T^B := \mathcal{M} \setminus (dom^{\mathcal{M}} \cup choice^{\mathcal{M}})$, where $dom^{\mathcal{M}}$ and $choice^{\mathcal{M}}$ are the extensions in $\mathcal{M}$ of the predicates dom and $choice$, is a minimal model of $\Pi_T^{\mathcal{M}_B}$. This is immediate by construction: the global atoms obtained by applying the rules

$$P_j(\bar{X}_j) \leftarrow V_i(\bar{X}), \quad \bigwedge_{Z_l \in (\bar{X}_j \setminus \bar{X})} F_i^l(\bar{X}, Z_l),$$

are exactly those we put into $\mathcal{M}$. Then, $\mathcal{M}_T^B$ is a minimal model of $\Pi_T^{\mathcal{M}_B}$. Furthermore, $D_{\mathcal{M}} = D$. □

Proof of Theorem 2: Each stable model of $\Pi(\mathcal{G})$ generates a legal instance. The only situation in which that instance may be non minimal instance occurs when the openness of a source is satisfied by another source or a set of sources. This can only happen if there is a way to generate a same global relation by more than one view, i.e. if there is a way to establish a mapping from the universal and existential variables to the constants and universal variables of other views.

We will prove that if, for a given stable model $\mathcal{M}$ of $\Pi(\mathcal{G})$, $D_{\mathcal{M}}$ is not minimal, then there is an admissible mapping h such that $h(S_i^l) \subseteq \bigcup_{S \in (\mathcal{S} \setminus \{S_i^l\})} L(S)$. We do this by contradiction, assuming there is no admissible mapping and that $D_{\mathcal{M}}$ is not minimal.

Under this assumption, there exists a stable model of $\Pi(\mathcal{G})$, $\mathcal{M}'$ such that $D_{\mathcal{M}'} \subsetneqq D_{\mathcal{M}}$. More specifically, there is a global relation atom, $P(\bar{a})$, that belongs to $\mathcal{M}$, but not to $\mathcal{M}'$.

The global relation P above cannot be a predicate without existential variables in every view definition where it appears in the body, otherwise its same atoms will appear in all the legal instances. So, P must appear with existential variables in at least one view definition.

Then for $P(\bar{a})$ there must be a view, say V_1, that generated this atom unnecessarily, because there was another atom $P(\bar{a}')$, generated by a view V_2, that had already forced the satisfaction of the openness of V_1.

Each of these global atoms has some attributes generated by the contents of the views that have P in their definitions, and others by Skolem functions. The first correspond to universal variables, and the second, to existential variables in the view definitions. We distinguish those attributes by using the following notation for an atom: $P(\bar{a}_u, \bar{a}_e)$, $P(\bar{a}'_u, \bar{a}'_e)$.

We will first restrict ourselves to a view that is defined only by the global relation P, i.e. $V_1(\bar{x}) \leftarrow P(\bar{x}, \bar{z})$. Since $P(\bar{a}'_u, \bar{a}'_e)$ satisfied the openness of V_1, it holds that $\bar{a}_u \subset \bar{a}'_u$. Assume that $P(\bar{a}_u, \bar{a}_e)$ and $P(\bar{a}'_u, \bar{a}'_e)$ have (were derived from) u_1 and u_2 many universal variables and e_1 and e_2 many existential variables. That is $|\bar{a}_u| = u_1, |\bar{a}'_u| = u_2, |\bar{a}_e| = e_1, |\bar{a}'_e| = e_2$. Now, there is a mapping h of the view section of V_1 that maps $P(\bar{a}_u, \bar{a}_e)$ to $P(\mu^{u_2}, \varepsilon^{e_2}) \subseteq L(P(\bar{a}'_u, \bar{a}'_e))$.[6] Then we have reached a contradiction and the theorem holds in this case.

The proof, in the case where the view V_1 contains more that one global relation in its definition, is similar, but more involved, but goes through making the following considerations. For the openness of a view to be satisfied by another view or set of other views, we have to consider the set of global relations with a common existential variable in its view. In this case, the attributes generated by the universal variables need to be a subset of the attributes generated by the universal variables of the other view or the other set of views. This implies that if $D_{\mathcal{M}}$ is not minimal, there is an admissible mapping and therefore we reach a contradiction as before. □

The following intermediate results refer to the refined program 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. Assume that $D_{\mathcal{M}}$ is not legal. Then there must be a view V_i, with definition $\varphi\colon\ V_i(\bar{x}) \leftarrow \bigwedge_{i=1}^{n} P_i(\bar{x}_i, \bar{z}_i)$, for which $v_i \not\subseteq \varphi(D_{\mathcal{M}})$. More specifically, there is $\bar{a}$ such that $\bar{a} \in (v_i \setminus \varphi(D_{\mathcal{M}}))$.

Since $\mathcal{M}$ satisfies rules 3.(a) and 5. of the program $\Pi(\mathcal{G})$ in Definition 10, the global predicates in φ that do not contain existential variables get their values from V_i. So, at least for one of the sections S_{ij} there is no set of atoms in $\mathcal{M}$ such

[6] Here, X^Y denotes that attribute X is repeated Y times.

that for a view $V_{S_{ij}}(\bar{X}') \leftarrow \bigwedge_{P_k \in S_{ij}} P_k(\bar{X})$, where $\bar{X}'$ are the universal variables of S_{ij}, it does not hold that $\bar{a}'$ is in the view.

Since the last three (sets of) rules in 3.(b) of $\Pi(\mathcal{G})$ are satisfied by $\mathcal{M}$ and the predicates of view V_i that belong to S_{ij} are not satisfied by other views (this is because S_{ij} is not satisfied by $\mathcal{M}$), we have $add_{v_{ij}}(\bar{a}) \in \mathcal{M}$. This will open two alternatives wrt the rules in 4.:

- If the atoms in the predicates containing an existential variable Z were already generated by other views, then the Skolem predicate $F(\bar{a}', Z)$ will not be generated. Then the global predicates associated to Z will not be added again by the first rule in 3.(b).
- If the extensions of the predicates associated to Z were already generated by other views, then $F(\bar{a}', z) \in \mathcal{M}$ for a constant of the domain. Then the atoms in the predicates associated to Z will be added to the model by the first rule in 3.(b).

Either way, the atoms in the global predicates in S_{ij} were added by other views or by view V_i to $\mathcal{M}$. This implies that for the view $V_{S_{ij}}(\bar{X}') \leftarrow \bigwedge_{P_k \in S_{ij}} P_k(\bar{X})$, we have that $\bar{a}'$ is in the view. We have a contradiction. □

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. The idea of proof is similar to the proof of (26) for the simple program, but its realization is more involved now due to the structural complexity of the refined program. We need to define a Herbrand structure -now also with extensions for all the new auxiliary predicates- 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 10 of $\Pi(\mathcal{G})$. We put the following facts into $\mathcal{M}$:

1. $P_k(\bar{a}, t_d)$ for every global atom $P_k(\bar{a}) \in D$.
2. $dom(a)$ for every constant $a \in \mathcal{U}$.
3. $V_i(\bar{a})$ whenever $V_i(\bar{a}) \in v_i$ for some source extension v_i in $\mathcal{G}$.
4. For every view (source) predicate V_i in the system with description $V_i(\bar{X}) \leftarrow P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n)$ and for every P_k with no existential variables, the facts $P_k(\bar{x}_k, t_o)$.
5. For every atom $P_k(\bar{a}_k) \in D$, where $P_k(\bar{a}_k, t_o) \notin \mathcal{M}$, we will check which views have P_k in its body and have a $\bar{a}_k$ in its view extension with $\bar{a} \subseteq \bar{a}_k$. 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[7], such that $P_k \in S_i^l$, define the following views:

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

[7] 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.

Let P be the result of instantiating these views over the atoms in D and the source extensions. We will define $S^{P_k} = \{S_i^l \mid P_k(\bar{a}_k, S_i^l) \in P\}$. A priority relation will be defined between the view sections of S^{P_k} as follows:

(a) Each view section $S_i^l \in S^{P_k}$ that does not have an admissible mapping such that $h(S_i^l) \subseteq \bigcup_{S \in (\mathcal{S} \setminus \{S_i^l\})} L(S)$, where $\mathcal{S}$ is the set of all the view sections of $\mathcal{G}$, has the first priority.

(b) Each view section $S_i^l \in S^{P_k}$ that does not have the first priority, and such that there is an admissible mapping for which $h(S_i^l) \subseteq \bigcup_{S \in (\mathcal{S} \setminus \{S_i^l\})} L(S)$, where $\mathcal{S}$ is the set of view sections not necessarily from S^{P_k} that are in the situation defined in (a), has the second priority.

(c) Each view section $S_i^l \in S^{P_k}$ that does not have the first or second priority, and such that there is an admissible mapping $h \neq L$ for which $h(S_i^l) \subseteq \bigcup_{S \in (\mathcal{S} \setminus \{S_i^l\})} L(S)$, where $\mathcal{S}$ is the set of view sections not necessarily from S^{P_k} that are in the situation defined in (b), has the third priority.

(d) Each view section $S_i^l \in S^{P_k}$ that does not have the first, second or third priority, and such that the admissible mapping $h = L$, i.e $L(S_i^l) \subsetneq \bigcup_{S \in (\mathcal{S} \setminus \{S_i^l\})} L(S)$ where $\mathcal{S}$ is a set of view sections that are not in the cases (a) and (b), has the fourth priority.

Let S_i^m be the the view section of S^{P_k} with the highest priority. If there are two sections with the highest priority choose any of them. There is just one S_{ij}[8] in $\mathcal{G}$ such that $S_{ij} \supseteq S_i^m$. For this set we have $P_k(\bar{a}_k, v_{ij}) \in \mathcal{M}$, $add_{v_{ij}}(\bar{a}') \in \mathcal{M}$, $V_i(\bar{a}) \in \mathcal{M}$, $aux_{v_{ij}}(\bar{a}') \notin \mathcal{M}$, $var_{v_{ij}z_l}(\bar{a}_{z_l}) \notin \mathcal{M}$, $aux_{v_{ij}Z_l}(\bar{a}') \notin \mathcal{M}$, $add_{v_{ij}z_l}(\bar{a}') \in \mathcal{M}$. For all the rest of the views of S^{P_k}, e.g. S_i^m, we have that the $var_{v_{in}z_m}(\bar{a}_{z_m}) \notin \mathcal{M}$.

6. For every $P_k(\bar{a}_k, v_{ij}) \in \mathcal{M}$, we add the fact $P_k(\bar{a}_k, nv_{km})$ to $\mathcal{M}$ for every $S_{km} \neq S_{ij}$.
7. For every $add_{v_{ij}}(\bar{a}'), add_{v_{ij}z_l}(\bar{a}'), P_k(\bar{a}_k, v_{ij}) \in \mathcal{M}$, add $F_i^l(\bar{X}, z_l)$ into $\mathcal{M}$, where z_l is the value of that existential variable in $P_k(\bar{a}_k, v_{ij})$.

By construction $\mathcal{M}$ is a minimal model $\Pi(\mathcal{G})^{\mathcal{M}}$ 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 that that $D_{\mathcal{M}}$ is not a minimal instance of $\mathcal{G}$. Then there must be a minimal instance D such that $D \subsetneq D_{\mathcal{M}}$. By Lemma 2 we have that there is a model $\mathcal{M}'$ such that $D_{\mathcal{M}'} = D$. Then, $D_{\mathcal{M}'} \subsetneq D_{\mathcal{M}}$. In particular, we have that there is an atom of a global relation, say $P_k(\bar{a}, t_d)$, such that $P_k(\bar{a}, t_d) \in \mathcal{M}$ and $P_k(\bar{a}, t_d) \notin \mathcal{M}'$. Since $P_k(\bar{a}, t_d) \notin \mathcal{M}'$ and since we could delete that tuple from $\mathcal{M}$ without needing

[8] Here the S_{ij} are those appearing in Definition 10.

to add another tuple $P_k(\bar{b}, t_d)$ to $\mathcal{M}'$, we have that $\mathcal{M}$ is not minimal and, in consequence, it cannot be a stable model. □

Proof of Theorem 3: Directly from Lemma 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,
   $P_k(\bar{X}_k, v_{ij}) \leftarrow add_{v_{ij}}(\bar{X}'),\ \bigwedge_{Z_l \in (\bar{X}_k \setminus \bar{X}')} F_i^l(\bar{X}', Z_l)$, for $P_k \in S_{ij}$.
   $add_{v_{ij}}(\bar{X}') \leftarrow V_i(\bar{X})$, where $\bar{X}' = \bar{X} \cap \{\bigcup_{P_k \in S_{ij}} X_k\}$.
4. For every predicate $F_i^l(\bar{X}', Z_l)$ introduced in 3.b., the rules,
   $F_i^l(\bar{X}', Z_l) \leftarrow add_{v_{ij}Z_l}(\bar{X}'), dom(Z_l), choice((\bar{X}'), (Z_l))$.
   $add_{v_{ij}Z_l}(\bar{X}') \leftarrow add_{v_{ij}}(\bar{X}')$, for $l = 1, \ldots m$.
5. For every global relation $P(\bar{X})$ the rules
   $P(\bar{X}, nv_{ij}) \leftarrow P(\bar{X}, v_{hk}),$ for $\{(ij, hk) | P(\bar{X}) \in S_{ij}$ and $S_{hk}\}$.
   $P(\bar{X}, nv_{ij}) \leftarrow P(\bar{X}, t_o),$ for $\{(ij) | P(\bar{X}) \in S_{ij}\}$.
   $P(\bar{X}, t_d) \leftarrow P(\bar{X}, v_{ij}),$ for $\{(ij) | P(\bar{X}) \in S_{ij}\}$.
   $P(\bar{X}, t_d) \leftarrow P(\bar{X}, t_o)$.

This is a positive program with choice. Because of the second rule in 3.(b) and the second rule in 4., we can replace every occurrence of $add_{v_{ij}}(\bar{X}')$ and $add_{v_{ij}Z_l}(\bar{X}')$ by $V_i(\bar{X})$. Also from the third and fourth rules in 5., we can replace every occurrence of $P(\bar{X}, t_o)$ 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 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,
   $$P_k(\bar{X}_k, t_d) \leftarrow V_i(\bar{X}), \ \bigwedge_{Z_l \in (\bar{X}_k \setminus \bar{X}')} F_i^l(\bar{X}', Z_l), \text{ for } P_k \in S_{ij}.$$
4. For every predicate $F_i^l(\bar{X}', Z_l)$ introduced in 3.b., the rules,
   $$F_i^l(\bar{X}', Z_l) \leftarrow V_i(\bar{X}), dom(Z_l), choice((\bar{X}'), (Z_l)).$$

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