Optimizing Repair Programs and their Evaluation for Consistent Query Answering in Databases

Monica Caniupan, **Leopoldo Bertossi**, and **Loreto Bravo**
Carleton University, School of Computer Science, Ottawa, Canada.
{mcaniupa,bertossi,lbravo}@scs.carleton.ca

Abstract. Databases may not satisfy integrity constraints (ICs) for several reasons. Nevertheless, in most of the cases an important part of the data is still consistent wrt certain desired ICs, and the database can still give some correct answers to queries wrt those ICs. Consistent query answers are characterized as ordinary answers obtained from every minimally repaired and consistent version of the database. Database repairs can be specified as stable models of disjunctive logic programs with program constraints. In this paper, we optimize repair programs, model computation, and query evaluation over them. We make repair programs more compact by eliminating redundant rules and programs denial constraints when possible. These results facilitate the application of magic sets techniques to query evaluation in general and in the DLV system in particular, a reasoning system for logic programs under stable model semantics. We also investigate the applicability of magic sets techniques to repair program for consistent query answering under the restrictions imposed by current implementations, like DLV. We also analyze the implementation in DLV of queries with aggregate functions.

1 Introduction

Integrity constraints play an important role in databases. They capture the intended meaning (semantics) of the data in the database. Nevertheless, databases may become inconsistent with respect to ICs due to several reasons, among them: (a) In virtual data integration [25] of multiple data sources, possibly individually consistent wrt local ICs, the system may become inconsistent wrt global ICs [15]. (b) A stand alone relational database management system may not have mechanisms to maintain certain ICs. (c) In legacy systems data may not satisfy new semantic constraints. (d) In the presence of user or informational constraints, which are used, but not necessarily enforced by the system.

Even though, ICs may be violated by databases, in most of the cases only a small portion of the data is inconsistent wrt those ICs. In consequence, it becomes necessary to develop methods for retrieving consistent answers to queries. The notion of consistent answers to first-order (FO) queries was initially defined in [1], together with a mechanism for computing them. Intuitively, a ground tuple $\bar{t}$ is a consistent answer to a query $Q(\bar{x})$ in a database instance DB, if it is an ordinary answer to $Q(\bar{x})$ in every minimal repair of DB, where a repair is a database instance obtained from DB by deleting or inserting tuples, that satisfies the ICs and differ minimally (under set inclusion) from DB.

The mechanism presented in [1] to compute consistent answers is based on first-order query rewriting. Basically, given a non-existentially quantified conjunctive query $\mathcal{Q}$, a new query is generated, such that, when posed to a database, its usual answers correspond to the consistent answers to $\mathcal{Q}$ wrt the ICs. That method works for a restricted set of ICs, such as functional dependencies, and full set inclusion dependencies. However, it does not consider queries or ICs with existential quantifiers, like referential ICs.

In [2, 4, 5] a more general approach based on logic programs with stable models semantics [20] was introduced. There, database repairs are specified as the stable models of a disjunctive program with program denial constraints[1]. Thus, there exists a one to one correspondence between the stable models of the logic program and the database repairs. The logical approach works for all universal ICs and FO queries. In [7] the methodology was extended to handle acyclic referential integrity constraints as well.

Example 1. The database instance $\{S(a)\}$ is inconsistent wrt the inclusion dependency $\forall\, x\;\; S(x) \rightarrow Q(x)$. Consistency can be minimally restored by inserting $Q(a)$ or eliminating $S(a)$. The repair program contains the following rules [7]:

1. $dom(a)$.
2. $S(a, \mathbf{t_d})$.
3. $S(x, \mathbf{f_a}) \vee Q(x, \mathbf{t_a}) \leftarrow S(x, \mathbf{t}^{\star}), Q(x, \mathbf{f_a}), dom(x)$.
4. $S(x, \mathbf{f_a}) \vee Q(x, \mathbf{t_a}) \leftarrow S(x, \mathbf{t}^{\star}), \mathit{not}\ Q(x, \mathbf{t_d}), dom(x)$.
5. $S(x, \mathbf{t}^{\star}) \leftarrow S(x, \mathbf{t_d}), dom(x)$. (similar for Q)
   $S(x, \mathbf{t}^{\star}) \leftarrow S(x, \mathbf{t_a}), dom(x)$. (similar for Q)
6. $S(x, \mathbf{t}^{\star\star}) \leftarrow S(x, \mathbf{t_a})$. (similar for Q)
   $S(x, \mathbf{t}^{\star\star}) \leftarrow S(x, \mathbf{t_d}), \mathit{not}\ S(x, \mathbf{f_a})$. (similar for Q)
7. $\leftarrow S(x, \mathbf{t_a}), S(x, \mathbf{f_a})$. $\leftarrow Q(x, \mathbf{t_a}), Q(x, \mathbf{f_a})$.

We can see that the repair programs use annotation constants in an extra argument, actually each atom $P(\bar{a})$ can receive one of the following constants (with the following intuitive meaning on the right):

$P(\bar{a}, \mathbf{t_d})$	$P(\bar{a})$ is true in the database.
$P(\bar{a}, \mathbf{t_a})$	$P(\bar{a})$ is advised to be made true.
$P(\bar{a}, \mathbf{f_a})$	$P(\bar{a})$ is advised to be made false.
$P(\bar{a}, \mathbf{t}^{\star})$	$P(\bar{a})$ is true or is made true.
$P(\bar{a}, \mathbf{t}^{\star\star})$	$P(\bar{a})$ is true in *the repair*.

Rules 1 and 2 capture the domain constants and database facts respectively. The most important rules are 3 and 4. They establish the form of repairing the database according to the inclusion dependency; i.e. by making $Q(x)$ true or $S(x)$ false. Rules 5 capture the atoms that become true in the program (which are annotated with $\mathbf{t}^{\star}$). Rules 6 capture the atoms that become true in the repairs (which are annotated with $\mathbf{t}^{\star\star}$). Rules 7 are

[1] Programs constraints are head-free rules; program denial constraints are program constraints with only positive and built-in atoms in the body. (Database) denial constraints are ICs, i.e. conditions that have to be satisfied by the database relations; that can be written as program denial constraints. However the role of a program constraint (denial or not) is to discard the stable models that violate them. In the following we will use "(denial) constraint" for the database case, and "program (denial) constraint" for programs.

program denial constraints, which ensure that models containing atoms with both $\mathbf{t_a}, \mathbf{f_a}$ constants will not be generated (an atom cannot become true and false at the same time). This program has two stable models, one corresponding to inserting $Q(a)$, and the other eliminating $S(a)$. □

In this paper we describe optimizations to the logic approach presented in [7], which can be classified into two groups:

- *Structure of repair programs*: this basically involves changing the program (while keeping the same repair semantics): elimination of redundant rules, auxiliaries predicates, and (some) annotations constants. In addition, we make an intelligent generation of program denial constraints, so that they are generated only when needed.

- *Evaluation of repair programs*: we can compute consistent answers by evaluating queries and repair programs in DLV system [27]. However, as we will see later, usually only a subset of the program and the database facts is needed to compute answers to a specific query. We explore the use of magic sets methodologies [6] to capture that subset. Magic sets optimize the bottom-up processing of queries by simulating a top-down evaluation of queries [9]. This allows to focalize in the relevant part of the program and database facts instead of considering the whole set of rules and set of facts. The set of rules and database facts that involve predicates and parameters related to the predicates and values in the query are taken into account. In particular, with magic sets only a relevant subset of the database will be used for query evaluation.

Through structural optimizations we get simpler repair programs, that are easier to evaluate by a reasoning system. The second set of optimizations makes query processing more efficient. In general consistent query answering over inconsistent databases is an expensive computational task, actually in the worst case, Π_2^P-complete in data complexity [8, 10], i.e. of the same data complexity as evaluation of general disjunctive logic programs under stable model semantics [12]. Speeding up query evaluation over large data sets becomes particularly relevant.

Magic sets methods, originally introduced for deductive databases and datalog programs [9], have been extended to logic programs with stable model semantics [16], and some of them have been implemented in DLV [11].

Optimizations on the process of retrieving consistent answers have been studied and introduced before in the context of data integration [15], where techniques to efficiently compute and store database repairs are described. Basically, database facts participating in violations of universal ICs are located and extracted from the database. This splits the database in two parts: the affected database, which contains data violating ICs; and the safe database, which stores consistent data. That operation permits to speed up the computation of database repairs, that are computed for the affected part. Nevertheless, our goal in this paper is not the computation of repairs, but the efficient computation of consistent answers. So that, our optimizations are conducted in that direction.

In this paper we also describe how to specify repair programs to be used to compute consistent answers to scalar aggregate queries, which were introduced in [3] using a *range semantics*, i.e. the answer to a query is an optimal interval that contains the value of the aggregate query in every possible repair. Here we use logic programs instead of conflict graph representations as in [3]. We show how to exploit the capabilities of the DLV system to compute aggregate functions (*min, max, count, times, sum*) over stable models [17].

This paper is structured as follows: in section 2 we recall basic concepts on databases and repair programs. In section 3 the structural optimizations performed on repair programs are presented. In section 4 a magic sets methodology for disjunctive repair programs with program denial constraints is described. We also specify how to use DLV with magic sets for this kind of programs. In section 5 the specification of repair programs to compute consistent answers to scalar aggregate queries is presented. Section 6 presents some final conclusions.

2 Preliminaries

A relational database schema is denoted by $\Sigma = (\mathcal{U}, \mathcal{R} \cup \mathcal{B})$ where $\mathcal{U}$ is the possibly infinite database domain, $\mathcal{R}$ is a set of database predicates, and $\mathcal{B}$ is a set of built-in predicates. Database instances of a relational schema are finite collections DB of ground atoms $P(c_1, ..., c_n)$, where P is a database predicate, and $c_1, ..., c_n$ are constants in the database domain $\mathcal{U}$. Extensions for built-in predicates are fixed in every database instance. There is also a fixed set of integrity constraints (IC) that are expected to be satisfied by any database instance, but they may not. We consider universal integrity constraints, and referential integrity constraints (RIC) [7].

A *universal integrity constraint* (UIC) is a any first-order (FO) sentence that is logically equivalent to a sentence of the form

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

where $\bar{\forall}$ is a prefix of universal quantifiers, $P_i, Q_j \in \mathcal{R}$, and φ is a formula containing built-in atoms from $\mathcal{B}$ only. A *referential integrity constraint* is a sentence of the form

$$\forall \bar{x} \ (P(\bar{x}) \rightarrow \exists y \ Q(\bar{x}', y)), \tag{2}$$

where $\bar{x}' \subseteq \bar{x}$ and $P, Q \in \mathcal{R}$.

Example 2. For schema $\{Student(id, name), Grad(id, name), Assistant(id, course)\}$ UICs are the functional dependency (FD) $Student : id \rightarrow name$, expressed in FO logic by $\forall \, id \, name_1 \, name_2 \ (Student(id, name_1) \wedge Student(id, name_2) \rightarrow name_1 = name_2)$; and the full inclusion dependency (IND) $Grad[id, name] \subseteq Student[id, name]$, expressed by $\forall \, id \, name \ (Grad(id, name) \rightarrow Student(id, name))$.
The inclusion dependency $Assistant[id] \subseteq Student[id]$ can be expressed as a RIC: $\forall \ id \ course \ (Assistant(id, course) \ \rightarrow \ \exists \ name \ Student(id, name))$. Here $\bar{x} = (id, course)$, $\bar{x}' = (id)$, and $\bar{y} = (name)$. $\Box$

A database instance DB is *consistent* if it satisfies the given set IC of ICs. Otherwise, it is *inconsistent* wrt IC. The semantics of constraint satisfaction adopted here is the same of the one defined in [7], according to which a UIC of the form (1) is satisfied if every ground tuple $P(\bar{a}) \in DB$ with $\bar{a} \in (\mathcal{U} - \{null\})$ holds the UIC. A RIC of the form (2) is satisfied if for all $P(\bar{a}) \in DB$, with $\bar{a} \in (\mathcal{U} - \{null\})$, there exists a tuple $\bar{b}$ of constants in $\mathcal{U}$ for which $Q(\bar{a}', \bar{b}) \in DB$. In other words, UICs are satisfied if for tuples with non-null values the UICs hold, and RICs are classically satisfied when universally quantified variables in (2) take values different from null, and existentially quantified variables taken any value.

When inconsistencies arise in a DB, consistency can be restored by deleting and/or inserting tuples. In this way, a repair is a new database instance with the same schema as DB that satisfies ICs and differs minimally, under set inclusion, from the DB [1].

Database repairs can be specified as stable models (SM) of *disjunctive logic programs* [21]. The idea behind is that, given an inconsistent database instance DB and a set of ICs IC, a disjunctive repair program $\Pi(DB, IC)$ is constructed, such that there is a one to one correspondence between the stable models of $\Pi(DB, IC)$ and the repairs of DB [4, 5]. Disjunctive rules express the options for insertions or deletions of tuples needed to restore consistency.

As mentioned before, repair programs use annotation constants, whose role is to enable the definition of atoms so that they can become true in the repairs (database facts or new insertions of atoms) or false (deletion) in order to satisfy the ICs. Annotations are performed according to the following sequential steps: first ground atoms $P(\bar{c})$ from the database receive an extra argument $\mathbf{t_d}$, as a consequence, $P(\bar{a}, \mathbf{t_d})$ becomes a fact in $\Pi(DB, IC)$. Then, for each IC a disjunctive rule is constructed in such a way that the body of the rule captures the violation condition for the IC; and the head describes how to restore the consistency by deleting or inserting the corresponding tuples. These endorsements are seized by the $\mathbf{t_a}, \mathbf{f_a}$ annotations. For instance, atom $P(\bar{a}, \mathbf{t_a})$ establishes the insertion of $P(\bar{a})$, and $P(\bar{a}, \mathbf{f_a})$ instances the deletion of $P(\bar{a})$. As an illustration, for the inclusion dependency $\forall\, x\ (S(x) \rightarrow Q(x))$, the disjunctive program rule:

$$S(x, \mathbf{f_a}) \vee Q(x, \mathbf{t_a}) \leftarrow S(x, \mathbf{t_d}),\ not\ Q(x, \mathbf{t_d}), \tag{3}$$

states that if the tuple $S(x, \mathbf{t_d})$ is a program fact but $Q(x, \mathbf{t_d})$ is not, then consistency is restored by deleting $S(x)$, which receives constant $\mathbf{f_a}$ in the head of the rule, or by inserting $Q(x)$, which receives the $\mathbf{t_a}$ constant.

The $\mathbf{t}^\star$ constant is introduced so as to keep repairing the database if there is interaction of ICs; then it becomes highly significant in cases where the insertion of a tuple may generate a new IC violation, e.g. if due to a different IC, $S(c, \mathbf{t_a})$ is generated but $Q(c)$ is not in the database, the constraint is once again violated. The aftermath is that the program rule (3) has to be changed to:

$$S(x, \mathbf{f_a}) \vee Q(x, \mathbf{t_a}) \leftarrow S(x, \mathbf{t}^\star),\ not\ Q(x, \mathbf{t_d}), \tag{4}$$

where the atom $S(x, \mathbf{t}^\star)$ becomes true if either $S(x, \mathbf{t_d})$ or $S(x, \mathbf{t_a})$ are true.

Finally, atoms with constant $\mathbf{t}^{\star\star}$ are the ones that become true in the repairs. They are use to read off the database atoms in the repairs. The following program was introduced in [5, 7].

Definition 1. [5, 7] The repair program $\Pi(DB, IC)$ for database instance DB and ICs IC is composed by the following rules:
1. $dom(a)$ for each constant $a \in (\mathcal{U} - \{null\})$.
2. $P(\bar{a}, \mathbf{t_d})$ for each atom $P(\bar{a}) \in DB$.
3. For every global universal IC of form (1) the set of clauses:

$$\bigvee_{i=1}^{n} P_i(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} Q_j(\bar{y}_j, \mathbf{t_a}) \;\leftarrow\; \bigwedge_{i=1}^{n} P_i(\bar{x}_i, \mathbf{t}^\star), \bigwedge_{Q_j \in Q'} Q_j(\bar{y}_j, \mathbf{f_a}), \bigwedge_{Q_k \in Q''} \mathit{not}\ Q_k(\bar{y}_k, \mathbf{t_d}), dom(\bar{x}), \bar{\varphi}.$$

for every set Q' and set Q'' such that $Q' \cup Q'' = \bigcup_{i=1}^{m} Q_i$ and $Q' \cap Q'' = \emptyset$, where $\bar{x}$ is the tuple of all variables appearing in database atoms in the rule, and $\bar{\varphi}$ is a conjunction of built-ins equivalent to the negation of φ.
4. For every referential IC of form (2) the clauses:

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

5. For each predicate $p \in R$ annotation clauses:
$P(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x}, \mathbf{t_d}),\ dom(\bar{x})$.
$P(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x}, \mathbf{t_a}),\ dom(\bar{x})$.
6. For every predicate $P \in \mathcal{R}$, the interpretation clauses:
$P(\bar{x}, \mathbf{t}^{\star\star}) \;\leftarrow\; P(\bar{x}, \mathbf{t_a})$.
$P(\bar{x}, \mathbf{t}^{\star\star}) \;\leftarrow\; P(\bar{x}, \mathbf{t_d}),\ \mathit{not}\ P(\bar{x}, \mathbf{f_a})$.
7. For every predicate $P \in \mathcal{R}$, the program denial constraint: $\leftarrow\; P(\bar{x}, \mathbf{t_a}), P(\bar{x}, \mathbf{f_a})$.
$\square$

Rules 1 captures the database constants, which are stored in an auxiliary predicate dom. Rules 2 establishes the program facts. Rules 3 and 4 capture the intended form of restoring consistency when ICs of the form (1), (2) are violated, respectively. Rules 5 captures the atoms that become true in the program. Rules 6 captures the atoms that become true in the repairs. Rule 7 represents the program denial constraints.

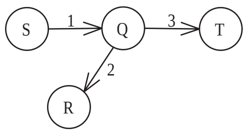
Database repairs are retrieved from the stable models of $\Pi(DB, IC)$: for each stable model $\mathcal{M}$ of $\Pi(DB, IC)$ a repair is generated by selecting the atoms with $\mathbf{t}^{\star\star}$ constant in $\mathcal{M}$.

Example 3. (example 1 cont.) $\Pi(DB, IC)$ for $DB = \{S(a)\}$ and IC: $\forall\, x\ (S(x) \rightarrow Q(x))$ is the program in example 1. It has the following stable models:
$\mathcal{M}_1 = \{dom(a), S(a, \mathbf{t_d}), S(a, \mathbf{t}^\star), \underline{S(a, \mathbf{t}^{\star\star})}, Q(a, \mathbf{t_a}), Q(a, \mathbf{t}^\star), \underline{Q(a, \mathbf{t}^{\star\star})}\}$;
$\mathcal{M}_2 = \{dom(a), S(a, \mathbf{t_d}), S(a, \mathbf{t}^\star), \overline{S(a, \mathbf{f_a})}\}$.
Thus, consistency is recovered by inserting $Q(a)$ ($Q(a, \mathbf{t}^{\star\star}) \in \mathcal{M}_1$) or deleting $S(a)$ ($S(a, \mathbf{f_a}) \in \mathcal{M}_2$). The repairs are $\{S(a), Q(a)\}$ and $\{\}$. $\square$

In [5] it was proved that the repair program of Definition 1 is a correct specification of database repairs wrt a *RIC-acylic* set of universal ICs of form (1) and of referential ICs of form (2). In order to define the *RIC-acyclic* property, we need to introduce some concepts:

Definition 2. The *dependency graph* $\mathcal{G}(IC)$ for a set of ICs IC of the form (1) or (2) is defined as follow: each database predicate P in DB is a node, and there is an edge (P_i, P_j) from P_i to P_j iff there exists a constraint $ic \in IC$ such that P_i appears in the antecedent of ic and P_j appears in the consequent of ic. In addition, there is an edge (P_i, P_i) if P_i appears in the antecedent of an ic which has only built-in predicates in its consequent. A node is called a *sink* (*source*) if it has only incoming (outgoing) edges. □

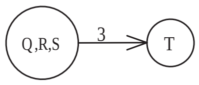
Example 4. Given a set IC of two universal integrity constraints $IC_1 = S(x) \rightarrow Q(x)$ and $IC_2 = Q(x) \rightarrow R(x)$, and a referential integrity constraint $IC_3 = Q(x) \rightarrow T(x,y)$, the dependency graph $\mathcal{G}(IC)$ is the following:



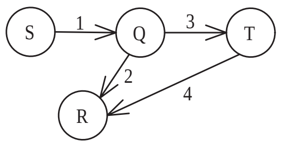
Edges 1 and 2 correspond to the universal constraints IC_1 and IC_2 resp., and edge 3 to the referential IC. Nodes R and T are sink nodes, S is a source node. □

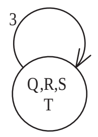
Definition 3. Given a set IC of UICs and RICs, IC_U denotes the set of UICs in IC. The *contracted dependency graph*, $\mathcal{G}^C(IC)$, of IC is obtained from $\mathcal{G}(IC)$ by replacing for every $c \in \mathcal{C}(\mathcal{G}(IC))$ the vertices in $\mathcal{V}(c)$ by a single vertex and deleting all the edges associated to the elements of IC_U. Finally, IC is said to be *RIC-acyclic* if $\mathcal{G}^C(IC)$ has no cycles. □

Example 5. (example 4 cont.) The contracted dependency graph $\mathcal{G}^c(IC)$ is obtained by replacing in $\mathcal{G}(IC)$ the edges 1 and 2 and their end vertices by a vertex labelled with $\{Q, R, S\}$.



Since there are no loops in $\mathcal{G}^c(IC)$, the set IC is RIC-acylic. If we add a new UIC: $T(x,y) \rightarrow R(y)$ to IC, all the vertices belong to the same connected component. $\mathcal{G}(IC)$ and $\mathcal{G}^c(IC)$ are, respectively:





Since there is a self-loop in $\mathcal{G}^c(IC)$, the set of ICs is *not* RIC-acylic. □

For RIC-acyclic integrity constraints, the repair program without the program denial constraints, is *locally stratified*.

Definition 4. [29] A program is locally stratified iff its herbrand base[2] can be partitioned into sets $S_0, S_1, \ldots$ (called strata) such that for each rule

$$A_1 \vee \cdots \vee A_l \leftarrow B_1, \ldots, B_m,\ not\ C_1, \ldots,\ not\ C_n \in ground(P)^3$$

[2] For any program P, the herbrand base is the set of all ground literals constructible from the predicate symbols appears in P and the constants of the herbrand universe. The latter, is the set of all constants appearing in P.

[3] $ground(P)$ denotes the set of ground rules of program P. A rule is ground if no variables appears in it.

there exists an $i \geq 1$ such that all $A_1, \ldots, A_l$ belong to S_i, all $B_1, \ldots, B_m$, belong to $S_0 \cup \cdots \cup S_i$, and all $C_1, \ldots,$ $not\ C_n$ belong to $S_0 \cup \cdots \cup S_{i-1}$. For such a partition, we use P_i to denote the set of all rules from $ground(P)$ whose consequent belong to S_i. $\square$

We will use this result later to prove that we can use the magic sets technique (section 4) in the evaluation of repair programs. It has been shown that magic sets considerably improve the evaluation of queries [11].

Proposition 1. For a database DB and a RIC-acyclic set of universal and referential integrity constraints IC, the repair program $\Pi(DB, IC)$ without the program denial constraints is locally stratified. $\square$

First order queries posed over inconsistent databases are translated into logic programs. Given a query $\mathcal{Q}$, a new query $\Pi(\mathcal{Q})$ is generated by first expressing it as a datalog program [28], and next changing every positive literal $P(\bar{s})$ by $P(\bar{s}, \mathbf{t}^{\star\star})$, and every negative literal by $not\ P(\bar{s}, \mathbf{t}^{\star\star})$. Thus, in order to get consistent answers, $\Pi(\mathcal{Q})$ is "run" together with the corresponding repair program $\Pi(DB, IC)$. So that, consistent query answering is translated to cautious reasoning in stable model semantics [20]. Note that for a FO query Q, program $\Pi(DB, IC) \cup \Pi(\mathcal{Q})$ is still locally stratified.

Example 6. (example 3 cont.) Given the datalog query $\mathcal{Q}\colon Ans(x) \leftarrow S(x)$, $\Pi(\mathcal{Q})$ is $Ans(x) \leftarrow S(x, \mathbf{t}^{\star\star})$. The stable models of $\Pi(DB, IC) \cup \Pi(\mathcal{Q})$ do not have S atoms annotated with $\mathbf{t}^{\star\star}$ in common, so there are no consistent answers to $\mathcal{Q}$. $\square$

It is possible to identify classes of ICs and queries for which data complexity is lower than the worst-case Π_2^P-complete data complexity. In [1] CQA wrt a set of universal ICs and projection free conjunctive queries is polynomial in data complexity. Other lower complexity classes for CQA are identified in [10, 19]. In [5] head-cycle free disjunctive repair programs are detected and translated into equivalent normal programs with lower computational complexity ($coNP$-complete) [12].

There are different repair policies in the literature: in [8] RICs are repaired by adding arbitrary elements of the domain, but in [10], by tuple deletions only. These and other alternative policies can be specified by repair programs.

3 Structural Optimizations of Repair Programs

The construction of repair programs and their evaluation can be improved by applying suitable structural modifications. For instance, in order to generate the program facts, and domain constants, it is necessary to process the whole database (which technically means bringing the database into main memory). So that, given a large set of data, the construction of programs can become a slow process.

In the next sections we describe how to eliminate annotations of database facts, rules for domain constants, and redundant rules. It is of particular interest the elimination of program denial constraints, because apart of eliminating unnecessary model checking, it allows for the application of magic sets as implemented in the DLV system (c.f. section 4). Moreover, classes of ICs are identified, for which repair programs do not contain program constraints at all. In those cases, we can apply magic sets techniques (see section 4) to the optimized evaluation of repair programs as implemented in DLV, which currently requires the absence of program constraints.

3.1 Database Facts Annotations, Auxiliary Predicates, and Redundant Rules

First, instead of manually inserting database facts into repair programs once annotated with the $\mathbf{t_d}$ constant, they are imported directly from the database, without any annotation; and that constant is eliminated from the programs. In consequence, the database predicate P and its version that becomes expanded with an extra argument for the annotation have to be told apart. For this, the latter are replaced by an underscored version of the predicate, e.g. $P(\bar{a}, \mathbf{t_a})$ becomes $P\_(\bar{a}, \mathbf{t_a})$, etc.

Furthermore, the auxiliary dom predicate is also eliminated. That predicate was introduced to extract database constants, that are useful to check satisfiability of ICs. Now, instead of checking that variables are restricted to the database domain, we check that variables do not take null values. This is achieved by adding in rules regarding to ICs only, conditions of the form $\bar{x} \neq null$, instead of using $dom(\bar{x})$. This does not change the semantics of repair programs, due to that the restriction over variables are kept in the rules that capture the ICs. For instance, the annotation rules (c.f. rules 5 in Definition 1) become: $P\_(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x})$, and $P\_(\bar{x}, \mathbf{t}^\star) \leftarrow P\_(\bar{x}, \mathbf{t_a})$, respectively.

Finally, instead of having two interpretation rules for each database predicate, only one rule will be used. So far, interpretation rules are (c.f. rules 6 in example 1): (i) $P\_(\bar{x}, \mathbf{t}^{\star\star}) \ \leftarrow \ P\_(\bar{x}, \mathbf{t_a})$ and (ii) $P\_(\bar{x}, \mathbf{t}^{\star\star}) \ \leftarrow \ P(\bar{x}),\ not\ P\_(\bar{x}, \mathbf{f_a})$. These rules define the atoms that become true in the repairs; which are the ones advised to be true (i) and, database facts that are not advised to be false (ii). Now, for each database predicate there is a unique interpretation rule, namely $P\_(\bar{x}, \mathbf{t}^{\star\star}) \ \leftarrow \ P\_(\bar{x}, \mathbf{t}^\star),\ not\ P\_(\bar{x}, \mathbf{f_a})$.

With these modifications database facts do not have to be preprocessed, and the number of rules in the repair programs decreases considerably.

3.2 Relevant Program Denial Constraints

Program denial constraints of repair program permit to discard models containing atoms $P(\bar{x})$ annotated with both $\mathbf{t_a}$ and $\mathbf{f_a}$. We can identify cases of ICs for which a repair program will never generate such models. In those cases, program constraints can be eliminated. Apart of eliminating unnecessary model checking, the elimination of program constraints allows for the application of magic sets as implemented in the DLV system (section 4).

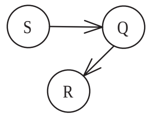
It can be seen that a repair program will have rules defining $P\_(\bar{x}, \mathbf{t_a})$, and $P\_(\bar{x}, \mathbf{f_a})$, for an atom $P(\bar{x})$, iff there exists at least two different ICs of the form (1) or (2) having $P(\bar{x})$ both in the antecedent of an IC and in the consequent of another. In those cases, program denial constraints for P should be kept.

Example 7. Given the $DB = \{S(a)\}$, and the set of ICs IC: $S(x) \rightarrow Q(x)$, and $Q(x) \rightarrow R(x)$, $\Pi(DB, IC)$ has the following rules for ICs and program denial constraints:
$S\_(x, \mathbf{f_a}) \vee Q\_(x, \mathbf{t_a}) \leftarrow S\_(x, \mathbf{t}^\star), Q\_(x, \mathbf{f_a}), x \neq null.$

$\underline{S}(x, \mathbf{f_a}) \vee \underline{Q}(x, \mathbf{t_a}) \leftarrow \underline{S}(x, \mathbf{t}^\star),\ not\ Q(x), x \neq null.$
$\underline{Q}(x, \mathbf{f_a}) \vee \underline{R}(x, \mathbf{t_a}) \leftarrow \underline{Q}(x, \mathbf{t}^\star), \underline{R}(x, \mathbf{f_a}), x \neq null.$
$\underline{Q}(x, \mathbf{f_a}) \vee \underline{R}(x, \mathbf{t_a}) \leftarrow \underline{Q}(x, \mathbf{t}^\star),\ not\ R(x), x \neq null.$
$\leftarrow \underline{Q}(x, \mathbf{t_a}), \underline{Q}(x, \mathbf{f_a}). \quad \leftarrow \underline{S}(x, \mathbf{t_a}), \underline{S}(x, \mathbf{f_a}). \quad \leftarrow \underline{R}(x, \mathbf{t_a}), \underline{R}(x, \mathbf{f_a}).$
In the case of predicate S (R) there is no way to generate an atom with constant $\mathbf{t_a}$ ($\mathbf{f_a}$ for R). Thus, the program denial constraints for S and R are always satisfied, and then, they can be eliminated. In contrast, for predicate Q both annotations are defined in the program, and then its program denial has to be kept. □

This idea can be formalized by appealing to the interaction between predicates as involved in ICs, which is capture for the dependency graph of Definition 2.

Example 8. (example 7 cont.) The figure shows the dependency graph $\mathcal{G}(IC)$ for IC: $S(x) \rightarrow Q(x)$, and $Q(x) \rightarrow R(x)$.



Node S is connected to Q due to the first IC, Q is connected to R due to the second IC. S is a source node, R is a sink node. □

Definition 5. Given a database instance DB, and a set of ICs IC, program $\Pi'(DB, IC)$ can be obtained from $\Pi(DB, IC)$ by deleting program denial constraints for the predicates that are sinks or sources in the corresponding dependency graph $\mathcal{G}(IC)$. □

Example 9. (example 7 and 8 cont.) Program $\Pi'(DB, IC)$ has the same set of rules as program $\Pi(DB, IC)$, except for the program constraints for the source predicate S and the sink predicate R in the dependency graph in example 8. □

Proposition 2. Given a database DB, and a set of ICs IC, $\Pi'(DB, IC)$ has the same stable models as $\Pi(DB, IC)$. □

Corollary 1. If the ICs are only formulas of the form $\bigwedge_{i=1}^{n} P_i\ (\bar{x}_i) \rightarrow \varphi$, where $P_i(\bar{x}_i)$ is an atom and φ is a formula containing built-ins, then the dependence graph $\mathcal{G}(IC)$ has only sink nodes. In consequence, the repair program $\Pi'(DB, IC)$ has no program denial constraints. □

This corollary include important classes of ICs such as, key constraints, functional dependencies, and range constraints.

By using all the transformations introduced so far, we obtain a new definition for the repair program.

Definition 6. Given a database instance DB, a set of ICs IC, the repair program $\Pi^\star(DB, IC)$ contains the set of rules:
1. $P(\bar{a})$ for each atom $P(\bar{a}) \in DB$.
2. For every global universal IC of form (1) the set of clauses:
$$\bigvee_{i=1}^{n} \underline{P}_i(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} \underline{Q}_j(\bar{y}_j, \mathbf{t_a}) \ \leftarrow\ \bigwedge_{i=1}^{n} \underline{P}_i(\bar{x}_i, \mathbf{t}^\star), \bigwedge_{Q_j \in Q'} \underline{Q}_j(\bar{y}_j, \mathbf{f_a}), \bigwedge_{Q_k \in Q''}\ not\ Q_k(\bar{y}_k), \bar{x} \neq null, \bar{\varphi}.$$

for every set Q' and set Q'' such that $Q' \cup Q'' = \bigcup_{i=1}^{m} Q_i$ and $Q' \cap Q'' = \emptyset$, where $\bar{x}$ is the tuple of all variables appearing in database atoms in the rule, and $\bar{\varphi}$ is a conjunction of built-ins equivalent to the negation of φ.
3. For every referential IC of form (2) the clauses:
$P\_(\bar{x}, \mathbf{f_a}) \vee Q\_(\bar{x}', null, \mathbf{t_a}) \leftarrow P\_(\bar{x}, \mathbf{t}^\star),\ not\ aux(\bar{x}'),\ not\ Q(\bar{x}', null), \bar{x} \neq null.$
$aux(\bar{x}') \leftarrow Q(\bar{x}', y),\ not\ Q\_(\bar{x}', y, \mathbf{f_a}),\ \bar{x}' \neq null, y \neq null.$
$aux(\bar{x}') \leftarrow Q\_(\bar{x}', y, \mathbf{t_a}),\ \bar{x}' \neq null, y \neq null.$
4. For each predicate $p \in R$ annotation clauses:
$P\_(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x}).$
$P\_(\bar{x}, \mathbf{t}^\star) \leftarrow P\_(\bar{x}, \mathbf{t_a}).$
5. For every predicate $P \in \mathcal{R}$, the interpretation clause:
$P\_(\bar{x}, \mathbf{t}^{\star\star}) \ \leftarrow\ P\_(\bar{x}, \mathbf{t}^\star),\ not\ P\_(\bar{x}, \mathbf{f_a}).$
6. For every predicate $P \in \mathcal{R}$ that is not a sink or a source node in $\mathcal{G}(IC)$, the program denial constraint: $\leftarrow\ P\_(\bar{x}, \mathbf{t_a}), P\_(\bar{x}, \mathbf{f_a}).$ □

Theorem 1. Given a database instance DB, and a set of ICs IC. The repair program $\Pi(DB, IC)$ as in Definition 1 and $\Pi^\star(DB, IC)$ produce the same repairs. □

Example 10. (example 7 cont.) $\Pi^\star(DB, IC)$ is composed by the rules:
Database facts: $S(a)$.
IC rules:
$S\_(x, \mathbf{f_a}) \vee Q\_(x, \mathbf{t_a}) \leftarrow S\_(x, \mathbf{t}^\star), Q\_(x, \mathbf{f_a}), x \neq null.$
$S\_(x, \mathbf{f_a}) \vee Q\_(x, \mathbf{t_a}) \leftarrow S\_(x, \mathbf{t}^\star),\ not\ Q(x), x \neq null.$
$Q\_(x, \mathbf{f_a}) \vee R\_(x, \mathbf{t_a}) \leftarrow Q\_(x, \mathbf{t}^\star), R\_(x, \mathbf{f_a}), x \neq null.$
$Q\_(x, \mathbf{f_a}) \vee R\_(x, \mathbf{t_a}) \leftarrow Q\_(x, \mathbf{t}^\star),\ not\ R(x), x \neq null.$
Annotation rules (similar for Q and R): $S\_(x, \mathbf{t}^\star) \leftarrow S(x).$ $S\_(x, \mathbf{t}^\star) \leftarrow S\_(x, \mathbf{t_a}).$
Interpretation rules (similar for Q and R): $S\_(x, \mathbf{t}^{\star\star}) \leftarrow S\_(x, \mathbf{t}^\star), not\ S\_(x, \mathbf{f_a}).$
Program denial constraints: $\leftarrow Q\_(x, \mathbf{t_a}), Q\_(x, \mathbf{f_a}).$
Stable models of program $\Pi^\star(DB, IC)$ contain less predicates than the models of $\Pi(DB, IC)$, that due to the fact that *dom* predicate was eliminated from repair programs. However they construct the same database repairs. DB becomes consistent by inserting the atoms $Q(a), R(a)$ ($\mathcal{M}_1$), or by deleting $S(a)$ ($\mathcal{M}_2$): $\mathcal{M}_1 = \{S(a),$ $S\_(a,\ \mathbf{t}^\star),\ Q\_(a,\ \mathbf{t_a}),\ \underline{S\_(a, \mathbf{t}^{\star\star})},\ Q\_(a,\ \mathbf{t}^\star),\ R\_(a,\ \mathbf{t_a}),\ \underline{Q\_(a, \mathbf{t}^{\star\star})},\ R\_(a,\ \mathbf{t}^\star),\ \underline{R\_(a, \mathbf{t}^{\star\star})}\}$, and $\mathcal{M}_2 = \{S(a), S\_(a, \mathbf{t}^\star), S\_(a, \mathbf{f_a})\}$. □

After applying all the optimizations presented so far, the program $\Pi^\star(DB, IC)$, for a database DB and a RIC-acyclic set of universal and referential ICs IC, is still locally stratified.

From now on, repair programs are those given in Definition 6, and they will be denoted just by $\Pi(DB, IC)$, as before.

4 Optimizing Query Evaluation

Consistent answers are obtained from stable models for the combination of the repair and query programs. Nevertheless, in most of the cases the former -so as its stable models- contain more information than necessary to answer the query, because repair programs are built considering all database predicates and database facts. However, query predicates are related to a subset of the database predicates. Furthermore, we are not interested in obtaining complete stable models (or repairs), but in only obtaining the consistent answer to our queries. In consequence, it is important to optimize the evaluation of the repair programs, considering only predicates and facts that are relevant to the query. This is precisely the purpose of the magic sets (MS) technique [6], that achieves it by simulating a top-down [9] -and then query directed- evaluation of the query through bottom-up propagation [9]. This technique produces a new program that contains a subset of the original rules, along with a set of new, "magic", rules.

Classic MS techniques for datalog programs [6, 30] have been extended to logic programs with unstratified negation under stable models semantics [16], to disjunctive logic programs with stratified negation [23], with an optimized version [11] being implemented in DLV. For this kind of programs, MS is sound and complete, i.e. the method computes all and only correct query answers for the query. In [24] a sound but incomplete methodology is presented for disjunctive programs with constraint rules of the form $\leftarrow C(\overline{x})$, where $C(\overline{x})$ is a conjunction of literals (a positive or negated atom). Here, we present a sound and complete methodology for our disjunctive repair programs with program denial constraints. The latter fall in the category of constraint rules with only positive intensional literals in the body. The methodology works for the kind of programs that we have, but not necessarily in the general case of disjunctive programs with constraints rules. It works as follows: the set of program denial constraints PD is separated from the rest of the rules, then MS, as defined in [11, 16], is applied to the resulting program. At the end of the process, the program constraints are put back into the resulting program, and so enforcing that the rewritten program has only coherent models.

4.1 Magic Sets For Repair Programs

Given an atomic ground (or partially grounded) query and a program, MS selects the relevant rules from the program to compute the answer for the query, and pushes down the query constants to restrict the tuples involved in the computation of the answer. Magic Sets carries this out by sequentially performing three well defined steps: *adornment, generation and modification*. The method will be illustrated using the following repair program and query, where rules have been enumerated to refer to them.

Example 11. Given a database instance $DB = \{S(a), T(a)\}$, and a set of ICs IC : $S(x) \rightarrow Q(x)$, $Q(x) \rightarrow R(x)$, and $T(x) \rightarrow W(x)$, $\Pi(DB, IC, \mathcal{Q}) := \Pi(DB, IC) \cup \Pi(\mathcal{Q})$ consists of the rules:

1. $S(a).\ T(a).$
2. $S\_(x, \mathbf{f_a}) \vee Q\_(x, \mathbf{t_a}) \leftarrow S\_(x, \mathbf{t^\star}), Q\_(x, \mathbf{f_a}),$ $x \neq null.$
3. $S\_(x, \mathbf{f_a}) \vee Q\_(x, \mathbf{t_a}) \leftarrow S\_(x, \mathbf{t^\star}),\ not\, Q(x),$ $x \neq null.$
4. $Q\_(x, \mathbf{f_a}) \vee R\_(x, \mathbf{t_a}) \leftarrow Q\_(x, \mathbf{t^\star}), R\_(x, \mathbf{f_a}),$ $x \neq null.$
5. $Q\_(x, \mathbf{f_a}) \vee R\_(x, \mathbf{t_a}) \leftarrow Q\_(x, \mathbf{t^\star}),\ not\, R(x),$ $x \neq null.$
6. $T\_(x, \mathbf{f_a}) \vee W\_(x, \mathbf{t_a}) \leftarrow T\_(x, \mathbf{t^\star}), W\_(x, \mathbf{f_a}),$ $x \neq null.$
7. $T\_(x, \mathbf{f_a}) \vee W\_(x, \mathbf{t_a}) \leftarrow T\_(x, \mathbf{t^\star}),\ not\, W(x),$ $x \neq null.$
8. $S\_(x, \mathbf{t^\star}) \leftarrow S\_(x, \mathbf{t_a}).$

9. $S\_(x,\mathbf{t}^\star) \leftarrow S(x)$.
10. $Q\_(x,\mathbf{t}^\star) \leftarrow Q\_(x,\mathbf{t_a})$.
11. $Q\_(x,\mathbf{t}^\star) \leftarrow Q(x)$.
12. $R\_(x,\mathbf{t}^\star) \leftarrow R\_(x,\mathbf{t_a})$.
13. $R\_(x,\mathbf{t}^\star) \leftarrow R(x)$.
14. $T\_(x,\mathbf{t}^\star) \leftarrow T\_(x,\mathbf{t_a})$.
15. $T\_(x,\mathbf{t}^\star) \leftarrow T(x)$.
16. $W\_(x,\mathbf{t}^\star) \leftarrow W\_(x,\mathbf{t_a})$.
17. $W\_(x,\mathbf{t}^\star) \leftarrow W(x)$.
18. $S\_(x,\mathbf{t}^{\star\star}) \leftarrow S\_(x,\mathbf{t}^\star), \mathit{not}\ S\_(x,\mathbf{f_a})$.
19. $Q\_(x,\mathbf{t}^{\star\star}) \leftarrow Q\_(x,\mathbf{t}^\star), \mathit{not}\ Q\_(x,\mathbf{f_a})$.
20. $R\_(x,\mathbf{t}^{\star\star}) \leftarrow R\_(x,\mathbf{t}^\star), \mathit{not}\ R\_(x,\mathbf{f_a})$.
21. $T\_(x,\mathbf{t}^{\star\star}) \leftarrow T\_(x,\mathbf{t}^\star), \mathit{not}\ T\_(x,\mathbf{f_a})$.
22. $W\_(x,\mathbf{t}^{\star\star}) \leftarrow W\_(x,\mathbf{t}^\star), \mathit{not}\ W\_(x,\mathbf{f_a})$.
23. $\leftarrow Q\_(x,\mathbf{t_a}), Q\_(x,\mathbf{f_a})$.
$\mathcal{Q}: Ans(x) \leftarrow S\_(x,\mathbf{t}^{\star\star})$.

The stables models are:

$$\begin{aligned}
\mathcal{M}_1 = \{&T(a), S(a), T\_(a,\mathbf{t}^\star), S\_(a,\mathbf{t}^\star), Q\_(a,\mathbf{t_a}), S\_(a,\mathbf{t}^{\star\star}), Q\_(a,\mathbf{t}^\star), R\_(a,\mathbf{t_a}), Q\_(a,\mathbf{t}^{\star\star}),\\
&R\_(a,\mathbf{t}^\star), R\_(a,\mathbf{t}^{\star\star}), \underline{Ans(a)}, W\_(a,\mathbf{t_a}), T\_(a,\mathbf{t}^{\star\star}), W\_(a,\mathbf{t}^\star), W\_(a,\mathbf{t}^{\star\star})\}\\
\mathcal{M}_2 = \{&T(a), S(a), T\_(a,\mathbf{t}^\star), S\_(a,\mathbf{t}^\star), Q\_(a,\mathbf{t_a}), S\_(a,\mathbf{t}^{\star\star}), Q\_(a,\mathbf{t}^\star), R\_(a,\mathbf{t_a}), Q\_(a,\mathbf{t}^{\star\star}),\\
&R\_(a,\mathbf{t}^\star), R\_(a,\mathbf{t}^{\star\star}), \underline{Ans(a)}, T\_(a,\mathbf{f_a})\}\\
\mathcal{M}_3 = \{&T(a), S(a), T\_(a,\mathbf{t}^\star), S\_(a,\mathbf{t}^\star), S\_(a,\mathbf{f_a}), W\_(a,\mathbf{t_a}), T\_(a,\mathbf{t}^{\star\star}), W\_(a,\mathbf{t}^\star), W\_(a,\mathbf{t}^{\star\star})\}\\
\mathcal{M}_4 = \{&T(a), S(a), T\_(a,\mathbf{t}^\star), S\_(a,\mathbf{t}^\star), S\_(a,\mathbf{f_a}), T\_(a,\mathbf{f_a})\}.
\end{aligned}$$

We can see that under cautious reasoning there are no answers to $\mathcal{Q}$. □

MS is applied over $\Pi^-(DB, IC, \mathcal{Q}) := \Pi(DB, IC, \mathcal{Q}) \smallsetminus PD$, where PD is the set of program denial constraints of program $\Pi(DB, IC, \mathcal{Q})$.

For the *adornment* step, the relationship between the query predicates and the predicates of the program Π^- are explicitly defined. The output of this step is a new *adorned* program denoted by $\mathcal{AP}(\Pi^-)$, where each intensional predicate (IDB) is of the form P^A, where A is a string of letters b, f (for *bound* and *free*, respectively) whose length is equal to the arity of predicate P.

Starting from the given query, adornments are created. First $\mathcal{Q}$ becomes $Ans^f(x) \leftarrow S\_^{fb}(x,\mathbf{t}^{\star\star})$, meaning that the first argument of $S\_$ is a free variable, and the second one is bound. The adorned predicate $S\_^{fb}$ is used to propagate bindings (adornments) onto the rules defining it. For instance, $S\_^{fb}$ propagate bindings to the rules 2, 3, 8, 9 and 18. Thus, the non-disjunctive rules 8 becomes $S\_^{fb}(x,\mathbf{t}^\star) \leftarrow S\_^{fb}(x,\mathbf{t_a})$. Rule 9 becomes $S\_^{fb}(x,\mathbf{t}^\star) \leftarrow S(x)$. Extensional predicates (EDB), i.e. facts as $S(x)$, only bind variables and do not receive any annotation. The same processing is done with rule 18.

For disjunctive rules, adorned predicates also propagates bindings to others head atoms in rules defining them. For instance, the adorned atom $S\_^{fb}$ used on rule 2: $S\_(x,\mathbf{f_a}) \vee Q\_(x,\mathbf{t_a}) \leftarrow S\_(x,\mathbf{t}^\star),\ Q\_(x,\mathbf{f_a}),\ x \neq \mathit{null}$, produces adornments over the head atom $Q\_(x,\mathbf{t_a})$, rule 2 becomes $S\_^{fb}(x,\mathbf{f_a}) \vee Q\_^{fb}(x,\mathbf{t_a}) \leftarrow S\_^{fb}(x,\mathbf{t}^\star), Q\_^{fb}(x,\mathbf{f_a})$, $x \neq \mathit{null}$. Rule 3 is now $S\_^{fb}(x,\mathbf{f_a}) \vee Q\_^{fb}(x,\mathbf{t_a}) \leftarrow S\_^{fb}(x,\mathbf{t}^\star),\ \mathit{not}\ Q(x),\ x \neq \mathit{null}$. Note that the adorned predicate $Q\_^{fb}$ also has to be processed. Finally, we generate the following adorned program $\mathcal{AP}(\Pi^-(DB, IC, \mathcal{Q}))$:

Program 1.
$Ans^f(x) \leftarrow S\_^{fb}(x,\mathbf{t}^{\star\star})$.
$S\_^{fb}(x,\mathbf{f_a}) \vee Q\_^{fb}(x,\mathbf{t_a}) \leftarrow S\_^{fb}(x,\mathbf{t}^\star), Q\_^{fb}(x,\mathbf{f_a}), x \neq \mathit{null}$.
$S\_^{fb}(x,\mathbf{f_a}) \vee Q\_^{fb}(x,\mathbf{t_a}) \leftarrow S\_^{fb}(x,\mathbf{t}^\star),\ \mathit{not}\ Q(x), x \neq \mathit{null}$.

$S\_^{fb}(x, \mathbf{t}^\star) \leftarrow S\_^{fb}(x, \mathbf{t_a}).$
$S\_^{fb}(x, \mathbf{t}^\star) \leftarrow S(x).$
$S\_^{fb}(x, \mathbf{t}^{\star\star}) \leftarrow S\_^{fb}(x, \mathbf{t}^\star), not\ S\_^{fb}(x, \mathbf{f_a}).$
$Q\_^{fb}(x, \mathbf{f_a}) \vee R\_^{fb}(x, \mathbf{t_a}) \leftarrow Q\_^{fb}(x, \mathbf{t}^\star), R\_^{fb}(x, \mathbf{f_a}), x \neq null.$
$Q\_^{fb}(x, \mathbf{f_a}) \vee R\_^{fb}(x, \mathbf{t_a}) \leftarrow Q\_^{fb}(x, \mathbf{t}^\star),\ not\ R(x), x \neq null.$
$Q\_^{fb}(x, \mathbf{t}^\star) \leftarrow Q\_^{fb}(x, \mathbf{t_a}).$
$Q\_^{fb}(x, \mathbf{t}^\star) \leftarrow Q(x).$
$Q\_^{fb}(x, \mathbf{t}^{\star\star}) \leftarrow Q\_^{fb}(x, \mathbf{t}^\star), not\ Q\_^{fb}(x, \mathbf{f_a}).$
$R\_^{fb}(x, \mathbf{t}^\star) \leftarrow R\_^{fb}(x, \mathbf{t_a}).$
$R\_^{fb}(x, \mathbf{t}^\star) \leftarrow R(x).$
$R\_^{fb}(x, \mathbf{t}^{\star\star}) \leftarrow R\_^{fb}(x, \mathbf{t}^\star), not\ R\_^{fb}(x, \mathbf{f_a}).$ □

Different strategies can be used when considering in what order atoms are to be processed and how bindings could be propagated. The process of passing bindings is called *sideways information passing strategies* (SIPS) [6]. Any of the SIPS strategies to be chosen have to ensure that all of the body atoms are processed. We follow the strategy adopted in [11] that is implemented in DLV, according to which only EDB predicates bind new variables, i.e. variables that do not carry a binding already. Furthermore, for disjunctive rules, head atoms that are different from the adorned atom that is being processed at a given point, only receive bindings, but do not bind any variable. For instance, the adorned atom $S\_^{bb}$ processed on rule $S\_(x, \mathbf{f_a}) \vee Q\_(x, y, \mathbf{t_a}) \leftarrow S\_(x, \mathbf{t}^\star),$ $Q\_(x, y, \mathbf{f_a}),\ x \neq null$; binds variable x of the head atom $Q\_(x, y, \mathbf{t_a})$, but y stays free.

The next step is the *generation of magic rules*; those that simulate a top-down evaluation of the query. They are generated for each rule of the adorned program. The generation differs for disjunctive and non-disjunctive adorned rules. In the latter case, for each adorned atom P^A in the body of an adorned rule, a magic rule is generated as follows: the head of the rule becomes the magic version of P^A, i.e. the new predicate $magic\_P^A$, from which all the arguments labelled with f in A are deleted. The body of the rule becomes the magic version of the adorned rule's head, followed by the corresponding predicates that are able to propagate the bindings on P^A. As an illustration, for the adorned rule $S\_^{fb}(x, \mathbf{t}^\star) \leftarrow S\_^{fb}(x, \mathbf{t_a})$, being $P^A = S\_^{fb}(x, \mathbf{t_a})$, the magic rule becomes: $magic\_S\_^{fb}(\mathbf{t_a}) \leftarrow magic\_S\_^{fb}(\mathbf{t}^\star)$.

In the case of disjunctive adorned rules, first intermediate non-disjunctive rule are generated, which is achieved by moving head atoms into the bodies of rules. Then, magic rules are generated as described previously. For instance, for the rule: $S\_^{fb}(x, \mathbf{f_a})$ $\vee\ Q\_^{fb}(x, \mathbf{t_a}) \leftarrow S\_^{fb}(x, \mathbf{t}^\star), Q\_^{fb}(x, \mathbf{f_a}), x \neq null$, two non-disjunctive rules are generated by changing S and Q predicates into the body. Thus constructing: $S\_^{fb}(x, \mathbf{f_a}) \leftarrow$ $Q\_^{fb}(x, \mathbf{t_a}), S\_^{fb}(x, \mathbf{t}^\star), Q\_^{fb}(x, \mathbf{f_a}), x \neq null$, and $Q\_^{fb}(x, \mathbf{t_a}) \leftarrow S\_^{fb}(x, \mathbf{f_a}), S\_^{fb}(x, \mathbf{t}^\star),$ $Q\_^{fb}(x, \mathbf{f_a}),\ x \neq null$. The following magic rules are constructed for the first non-disjunctive rule: $magic\_Q\_^{fb}(\mathbf{t_a}) \leftarrow magic\_S\_^{fb}(\mathbf{f_a})$; $magic\_S\_^{fb}(\mathbf{t}^\star) \leftarrow magic\_S\_^{fb}(\mathbf{f_a})$; $magic\_Q\_^{fb}(\mathbf{f_a}) \leftarrow magic\_S\_^{fb}(\mathbf{f_a})$. The set of magic rules is denoted by $\mathcal{MA}(\Pi^-)$. The magic rules $\mathcal{MA}(\Pi^-(DB, IC, \mathcal{Q}))$ for the adorned program 1 are:

Program 2.

$$\begin{array}{ll}
magic\_S\_^{fb}(\mathbf{t}^{\star\star}) \leftarrow magic\_ans^{f}. & magic\_Q\_^{fb}(\mathbf{t_a}) \leftarrow magic\_S\_^{fb}(\mathbf{f_a}). \\
magic\_S\_^{fb}(\mathbf{t}^{\star}) \leftarrow magic\_S\_^{fb}(\mathbf{f_a}). & magic\_Q\_^{fb}(\mathbf{f_a}) \leftarrow magic\_S\_^{fb}(\mathbf{f_a}). \\
magic\_S\_^{fb}(\mathbf{f_a}) \leftarrow magic\_Q\_^{fb}(\mathbf{t_a}). & magic\_S\_^{fb}(\mathbf{t}^{\star}) \leftarrow magic\_Q\_^{fb}(\mathbf{t_a}). \\
magic\_Q\_^{fb}(\mathbf{f_a}) \leftarrow magic\_Q\_^{fb}(\mathbf{t_a}). & magic\_S\_^{fb}(\mathbf{t_a}) \leftarrow magic\_S\_^{fb}(\mathbf{t}^{\star}). \\
magic\_Q\_^{fb}(\mathbf{t_a}) \leftarrow magic\_Q\_^{fb}(\mathbf{t}^{\star}). & magic\_S\_^{fb}(\mathbf{t}^{\star}) \leftarrow magic\_S\_^{fb}(\mathbf{t}^{\star\star}). \\
magic\_S\_^{fb}(\mathbf{f_a}) \leftarrow magic\_S\_^{fb}(\mathbf{t}^{\star\star}). & magic\_Q\_^{fb}(\mathbf{t}^{\star}) \leftarrow magic\_Q\_^{fb}(\mathbf{t}^{\star\star}). \\
magic\_Q\_^{fb}(\mathbf{f_a}) \leftarrow magic\_Q\_^{fb}(\mathbf{t}^{\star\star}). & magic\_R\_^{fb}(\mathbf{t_a}) \leftarrow magic\_Q\_^{fb}(\mathbf{f_a}). \\
magic\_Q\_^{fb}(\mathbf{t}^{\star}) \leftarrow magic\_Q\_^{fb}(\mathbf{f_a}). & magic\_R\_^{fb}(\mathbf{f_a}) \leftarrow magic\_Q\_^{fb}(\mathbf{f_a}). \\
magic\_Q\_^{fb}(\mathbf{f_a}) \leftarrow magic\_R\_^{fb}(\mathbf{t_a}). & magic\_Q\_^{fb}(\mathbf{t}^{\star}) \leftarrow magic\_R\_^{fb}(\mathbf{t_a}). \\
magic\_R\_^{fb}(\mathbf{f_a}) \leftarrow magic\_R\_^{fb}(\mathbf{t_a}). & magic\_R\_^{fb}(\mathbf{t_a}) \leftarrow magic\_R\_^{fb}(\mathbf{t}^{\star}). \\
magic\_R\_^{fb}(\mathbf{t}^{\star}) \leftarrow magic\_R\_^{fb}(\mathbf{t}^{\star\star}). & magic\_R\_^{fb}(\mathbf{f_a}) \leftarrow magic\_R\_^{fb}(\mathbf{t}^{\star\star}). \ \square
\end{array}$$

The last phase is the *modification step*, where magic atoms constructed in the generation stage are included in the body of adorned rules. Thus, for each adorned rule, the magic version of its head is inserted into the body. The rest of the adornments of the rule are now deleted. The set of modified rules is denoted by $\mathcal{MO}(\Pi^-)$. So that, modified rules $\mathcal{MO}(\Pi^-(DB, IC, \mathcal{Q}))$ for adorned program 1 are:

Program 3.

$Ans(x) \leftarrow magic\_Ans^f, S\_(x, \mathbf{t}^{\star\star}).$

$S\_(x, \mathbf{f_a}) \vee Q\_(x, \mathbf{t_a}) \leftarrow magic\_S\_^{fb}(\mathbf{f_a}), magic\_Q\_^{fb}(\mathbf{t_a}), S\_(x, \mathbf{t}^{\star}), Q\_(x, \mathbf{f_a}), x \neq null.$

$S\_(x, \mathbf{f_a}) \vee Q\_(x, \mathbf{t_a}) \leftarrow magic\_S\_^{fb}(\mathbf{f_a}), magic\_Q\_^{fb}(\mathbf{t_a}), S\_(x, \mathbf{t}^{\star}),\ not\ Q(x), x \neq null.$

$Q\_(x, \mathbf{f_a}) \vee R\_(x, \mathbf{t_a}) \leftarrow magic\_Q\_^{fb}(\mathbf{f_a}), magic\_R\_^{fb}(\mathbf{t_a}), Q\_(x, \mathbf{t}^{\star}), R\_(x, \mathbf{f_a}), x \neq null.$

$Q\_(x, \mathbf{f_a}) \vee R\_(x, \mathbf{t_a}) \leftarrow magic\_Q\_^{fb}(\mathbf{f_a}), magic\_R\_^{fb}(\mathbf{t_a}), Q\_(x, \mathbf{t}^{\star}),\ not\ R(x), x \neq null.$

$S\_(x, \mathbf{t}^{\star}) \leftarrow magic\_S\_^{fb}(\mathbf{t}^{\star}), S\_(x, \mathbf{t_a}).$

$S\_(x, \mathbf{t}^{\star}) \leftarrow magic\_S\_^{fb}(\mathbf{t}^{\star}), S(x).$

$Q\_(x, \mathbf{t}^{\star}) \leftarrow magic\_Q\_^{fb}(\mathbf{t}^{\star}), Q\_(x, \mathbf{t_a}).$

$Q\_(x, \mathbf{t}^{\star}) \leftarrow magic\_Q\_^{fb}(\mathbf{t}^{\star}), Q(x).$

$R\_(x, \mathbf{t}^{\star}) \leftarrow magic\_R\_^{fb}(\mathbf{t}^{\star}), R\_(x, \mathbf{t_a}).$

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

$S\_(x, \mathbf{t}^{\star\star}) \leftarrow magic\_S\_^{fb}(\mathbf{t}^{\star\star}), S\_(x, \mathbf{t}^{\star}), not\ S\_(x, \mathbf{f_a}).$

$Q\_(x, \mathbf{t}^{\star\star}) \leftarrow magic\_Q\_^{fb}(\mathbf{t}^{\star\star}), Q\_(x, \mathbf{t}^{\star}), not\ Q\_(x, \mathbf{f_a}).$

$R\_(x, \mathbf{t}^{\star\star}) \leftarrow magic\_R\_^{fb}(\mathbf{t}^{\star\star}), R\_(x, \mathbf{t}^{\star}), not\ R\_(x, \mathbf{f_a}).$ $\square$

The output of magic sets is the program $\mathcal{MS}(\Pi^-(DB, IC, \mathcal{Q}))$ that consist of the *magic rules* $\mathcal{MA}(\Pi^-(DB, IC, \mathcal{Q}))$, and the *modified rules* $\mathcal{MO}(\Pi^-(DB, IC, \mathcal{Q}))$. The final rewritten program denoted by $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$ consists of program $\mathcal{MS}(\Pi^-(DB, IC, \mathcal{Q}))$, the set of program denial constraints PD, and the *magic seed atom*, which is the magic version of the Ans predicate from the adorned query rule.

The rewritten version of the program in example 11, $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$, consists of the rules in program 2, the rules in program 3, $PD: \leftarrow Q\_(x, \mathbf{t_a}), Q\_(x, \mathbf{f_a})$ and the magic seed atom $magic\_Ans^f$. Program $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$ has the following two stable models:

$$\begin{aligned}
\mathcal{M}_1 = \{ & magic\_Ans^f, \underline{S(a)}, \underline{T(a)}, \underline{S\_(a, \mathbf{t}^{\star})}, magic\_S\_^{fb}(\mathbf{t}^{\star\star}), magic\_S\_^{fb}(\mathbf{f_a}), magic\_S\_^{fb}(\mathbf{t_a}), \\
& magic\_S\_^{fb}(\mathbf{t}^{\star}), magic\_Q\_^{fb}(\mathbf{f_a}), magic\_Q\_^{fb}(\mathbf{t_a}), magic\_Q\_^{fb}(\mathbf{t}^{\star}), magic\_R\_^{fb}(\mathbf{f_a}), \\
& magic\_R\_^{fb}(\mathbf{t_a}), \underline{Q\_(a, \mathbf{t_a})}, \underline{S\_(a, \mathbf{t}^{\star\star})}, \underline{Q\_(a, \mathbf{t}^{\star})}, \underline{R\_(a, \mathbf{t_a})}, \underline{Ans(a)}\}
\end{aligned}$$

$$\mathcal{M}_2 = \{magic\_Ans^f, \underline{S(a)}, \underline{T(a)}, \underline{S(a,\mathbf{t}^\star)}, magic\_S\_^{fb}(\mathbf{t}^{\star\star}), magic\_S\_^{fb}(\mathbf{f_a}), magic\_S\_^{fb}(\mathbf{t_a}), magic\_S\_^{fb}(\mathbf{t}^\star), magic\_Q\_^{fb}(\mathbf{f_a}), magic\_Q\_^{fb}(\mathbf{t_a}), magic\_Q\_^{fb}(\mathbf{t}^\star), magic\_R\_^{fb}(\mathbf{f_a}), magic\_R\_^{fb}(\mathbf{t_a}), \underline{S(a,\mathbf{f_a})}\}$$

Since there are no ground Ans atoms in common, from the original program there are no answers to $\mathcal{Q} : Ans(x) \leftarrow S(x, \mathbf{t}^{\star\star})$, which is now expressed as $Ans(x) \leftarrow magic\_Ans^f, S(x, \mathbf{t}^{\star\star})$ in the rewritten program. Nevertheless, for the rewritten program only models that are relevant to answer the query are computed. Furthermore, they are partially computed, i.e. they can be extended to stable models of the original program. Of course, without considering the magic predicates, which are auxiliary predicates that direct the course of query evaluation. For instance, model $\mathcal{M}_1$ of program $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$ is a subset of the stable models $\mathcal{M}_1$ and $\mathcal{M}_2$ in example 11; and model $\mathcal{M}_3$ is a subset of stable models $\mathcal{M}_3$ and $\mathcal{M}_4$ (without considering magic predicates). Instead of having four stable models as the original program, the rewritten program has only two stable models. In addition, the unique database predicates that are instantiated are the ones related to the query, i.e. Q, and R, in this case via the ICs. For the same reason, program $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$ contains rules related to predicates $\{S, Q, R\}$ of the original repair program (plus the magic rules), but not rules for predicates $\{T, W\}$, which are not relevant to the query.

Even though, in [24] it was shown that for general disjunctive programs with program constraints, MS does not always produce an equivalent rewritten program (completeness may be lost), we claim that for disjunctive repair programs with program denial constraints as in Definition 6, this MS methodology is both sound and complete. As a consequence, the rewritten program $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$, and the original repair program $\Pi(DB, IC, \mathcal{Q})$ are query equivalent in both brave and cautious reasoning[4].

Theorem 2. Given a database instance DB, a set of ICs IC, an atomic and possibly partially ground query $\mathcal{Q}$, program $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q})) \equiv_{\mathcal{Q}} \Pi(DB, IC, \mathcal{Q})$ under both the brave and cautious semantics. $\Box$

This result establishes that MS techniques are a good option for evaluating repair programs over large databases. In [11, 24] important results of the application of MS in evaluation of benchmark programs are reported.

This methodology based on leaving aside the program denial constraints when MS is applied and adding them at the end always works in the case of repair programs. This is because of two things. First, the rewritten program $\mathcal{MS}(\Pi^-(DB, IC, \mathcal{Q}))$ produced by MS contains all the rules that are necessary to check the satisfiability of the program constraints that are relevant to the query, in the sense that they contain predicates that are connected to the query predicate in the graph $\mathcal{G}(IC)$. More precisely, we have program denials of the form $\leftarrow P(\bar{x}, \mathbf{t_a}), P(\bar{x}, \mathbf{f_a})$ in $\Pi(DB, IC)$ only when there are rules defining $P(\bar{x}, \mathbf{t_a})$ and $P(\bar{x}, \mathbf{f_a})$ in $\Pi(DB, IC)$. In $\mathcal{MS}(\Pi^-(DB, IC, \mathcal{Q}))$, the output

[4] Programs Π_1 and Π_2 are bravely (resp. cautiously) equivalent w.r.t. a query Q, denoted $\Pi_1 \equiv_{\mathcal{Q}} \Pi_2$, if for any set F of facts, brave (resp. cautious) answers to Q from the program $\Pi_1 \cup F$ are the same as the brave (resp. cautious) answers to Q from $\Pi_2 \cup F$.

of the MS, we will still find all the rules defining P; then it will be possible to check the satisfiability of the program constraints in the models of $\mathcal{MS}(\Pi^{-}(DB, IC, \mathcal{Q}))$. Second, with MS we obtain a “subset”(without considering the magic atoms) of the stable models of the original program.

By the first remark, the stable models of the MS program satisfy the program denials. Furthermore, each of these models of the MS program contain limited extensions for database predicates (including annotations), those that are sufficient to answer the query as well, however each of them can be extended to a stable model of the original program. More precisely, it is possible to prove that, for every stable model M of $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$ (without considering the magic atoms), there is a stable model M' of $\Pi(DB, IC, \mathcal{Q})$ that extends M in the sense that $M = M$”, where M” is the set of atoms of M' that appear in the head of a rule in (the ground version of) $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$. As a consequence, the stable models of program $\mathcal{MS}^{\leftarrow}(\Pi(DB, IC, \mathcal{Q}))$ are all coherent models, they contain all the atoms needed to answer a query, and they compute the same answers as the models of program $\Pi(DB, IC, \mathcal{Q})$ for the given query.

Our approach works for all our repair programs, but it will not necessarily work for a more general disjunctive logic program. Sometimes, even if MS is applied to a disjunctive program with constraints that does not have stable models, the method can produce a program with stable models. This might happen if the query is related with a part of the program which is consistent wrt the program denial; and MS focalizes on that part of the program to answer the query. In addition, our MS method might not work for logic programs that do have stable models. For instance, for the database $\{R(a)\}$ and program Π with rules: $(Y(x) \leftarrow S(x))$, $(P(x) \leftarrow R(x),\ not\ S(X))$, $(S(x) \leftarrow R(x),\ not\ P(X))$, and program constraint $\leftarrow Y(x)$, we have one stable model $\mathcal{M} = \{R(a), P(a)\}$. But for query $Ans(x) \leftarrow P(x)$, our MS method produces a program that has two stable models (shown here without magic atoms): $\mathcal{M}_1 = \{R(a), P(a), \underline{Ans(a)}\}$; and $\mathcal{M}_2 = \{R(a), S(a)\}$, and as a consequence there are no cautious answers to the query even though a should be an answer to it. This happens because the MS method does not select rule $Y(x) \leftarrow S(x)$. Then, when the constraint $\leftarrow Y(x)$, is put back to the program it is satisfied even though it should not be. In our case, when we deal with repair programs, a rule that is relevant to check the satisfiability of a program denial constraint is never left out of the MS rewritten program.

4.2 Applying Magic Sets to CQA in DLV System

In theory magic sets can be successfully applied to the evaluation of disjunctive repair programs with program denial constraints. However, we do not know of any system that incorporates this technique for this kind of programs. There are no implementations that allow program constraints. Nevertheless, DLV does implement MS for disjunctive programs without program or denial constraints [27]. In this case, DLV applies MS internally, without giving access to the rewritten program (to which it would be easy to add the program denial constraints at the end). As a consequence, the application of MS with DLV for the evaluation of repair programs with program constraints is not straightforward. In this section we describe how to modify repair programs in order to be able to apply MS directly through DLV. Basically, program constraints are rewritten in such a way that DLV does not recognize them as denial rules, but it is still able to consider only coherent models, i.e. models without a same database atom annotated both with both $\mathbf{t_a}$ and $\mathbf{f_a}$.

Definition 7. Given a database instance DB, and a set of ICs IC, program $\Pi''(DB, IC)$ is obtained from $\Pi(DB, IC)$ by replacing each program denial constraint in it of the form $\leftarrow P\_(\bar{x}, \mathbf{t_a}),\ P\_(\bar{x}, \mathbf{f_a})$ by the rule $\ inc \leftarrow P\_(\bar{x}, \mathbf{t_a}),\ P\_(\bar{x}, \mathbf{f_a})$. □

Program $\Pi''(DB, IC)$ may have stable models that are not coherent models of the original program; namely those that contain both $P(\bar{c}, \mathbf{t_a})$ and $P(\bar{c}, \mathbf{f_a})$ for a given predicate P in DB and a constant c.

Example 12. (example 11 cont.) Program $\Pi(DB, IC)$ has one program denial constraint, for predicate Q. So that, $\Pi''(DB, IC)$ contains the modified denial rule $inc \leftarrow Q\_(x, \mathbf{t_a}), Q\_(x, \mathbf{f_a})$, and has two additional stable models:

$$\mathcal{M}_1 = \{T(a), S(a), T\_(a, \mathbf{t}^\star), S\_(a, \mathbf{t}^\star), \underline{Q\_(a, \mathbf{t_a})}, S\_(a, \mathbf{t}^{\star\star}), Q\_(a, \mathbf{t}^\star), \underline{Q\_(a, \mathbf{f_a})}, W\_(a, \mathbf{t_a}), \underline{inc}, T\_(a, \mathbf{t}^{\star\star}), W\_(a, \mathbf{t}^\star), W\_(a, \mathbf{t}^{\star\star})\};$$
$$\mathcal{M}_2 = \{T(a), S(a), T\_(a, \mathbf{t}^\star), S\_(a, \mathbf{t}^\star), \underline{Q\_(a, \mathbf{t_a})}, S\_(a, \mathbf{t}^{\star\star}), Q\_(a, \mathbf{t}^\star), \underline{Q\_(a, \mathbf{f_a})}, T\_(a, \mathbf{f_a}), \underline{inc}\}.$$

□

In order to retrieve consistent answers, a ground query $\mathcal{Q}$ has to be translated into $\mathcal{Q} \vee inc$, which does not affect the cautious semantics (being true in all stable models) and does not requires to discard the incoherent models. This is due to the fact that coherent models do not satisfy atom inc, and then they are required to satisfy $\mathcal{Q}$.

Example 13. (example 12 cont.) The query $\mathcal{Q}:\ (not\ S(a) \vee R(a))$ expressed as a program contains two rules: $Ans \leftarrow not\ S(a)$ and $Ans \leftarrow R(a)$. It is true in program $\Pi(DB, IC, \mathcal{Q})$, but becomes false when evaluated in $\Pi''(DB, IC, \mathcal{Q})$. However, the query $(not\ S(a) \vee R(a) \vee inc)$, which as a program is: $Ans \leftarrow not\ S(a)$, $Ans \leftarrow R(a)$, and $Ans \leftarrow inc$, is true when evaluated in $\Pi''(DB, IC, \mathcal{Q})$. □

The case of queries with variables, such that $Ans(x) \leftarrow S(x)$ is a bit different. It cannot be transformed into $Ans(x) \leftarrow S(x) \vee inc$, due to the fact that consistent answers are those contained in the intersection of all the coherent stable models of the program, i.e. they are based on cautious or skeptical reasoning. Clearly, incoherent models, those satisfying inc, when intersected with the coherent ones, could make us loose cautious answers; so, for incoherent models, we need to make extensions of the Ans predicate sufficiently large. A way to solve this problem is as follows: insert rules of the form $Ans(x) \leftarrow inc, P\_(x, \mathbf{t}^\star)$ into the query program, for each predicate P that is connected[5] to a query predicate in the graph $\mathcal{G}(IC)$. Atoms of the form $P\_(x, \mathbf{t}^\star)$ are those that become true during the repair process. Then, they are the ones that allow to give larger extension to the Ans predicate.

As an illustration, for query $Ans(x) \leftarrow S(x)$ evaluated with program in example 11, the following query rules have to be added: $Ans(x) \leftarrow inc, S\_(x, \mathbf{t}^\star)$, $Ans(x) \leftarrow inc, Q\_(x, \mathbf{t}^\star)$, $Ans(x) \leftarrow inc, R\_(x, \mathbf{t}^\star)$.

[5] A pair of nodes are connected in the graph $\mathcal{G}(IC)$ if there is a sequence of consecutive edges (consider as undirected edges) connecting vertices.

Proposition 3. Given a database instance DB, a set of ICs IC, and query program $\Pi(\mathcal{Q})$, if $\Pi''(\mathcal{Q})$ is obtained from $\Pi(\mathcal{Q})$ by adding rules on it of the form $Ans(x) \leftarrow inc, P\_(x, \mathbf{t}^\star)$, for each predicate P that is connected in $\mathcal{G}(IC)$ to a query predicate appearing in $\Pi(\mathcal{Q})$, then $\Pi''(DB, IC) \cup \Pi''(\mathcal{Q}) \equiv_{\mathcal{Q}} \Pi(DB, IC) \cup \Pi(\mathcal{Q})$ under the cautious semantics. $\square$

4.3 Selecting Relevant Database Facts

The repair programs in Definition 6 consider all the database facts (rule 1 in it), and all of them appear in the stable models of the program, even if they are not involved in the computation of answers to a particular query. In example 11 tuples regarding to predicate T appear in stable models, but they are not related to the predicate S in the query. In spite of this, the set of tuples that are relevant to answer a query can be selected before processing the query. This can be achieved by analyzing the relation between database predicates and query predicates as captured by the dependency graph in Definition 2. Intuitively, the database facts that are necessary to answer a query Q are among those that are associated to predicates connected in the graph to the predicates in the query.

Let $\Pi(DB, IC, \mathcal{Q}){\downarrow}\mathcal{Q}$ be the same as program $\Pi(DB, IC, \mathcal{Q})$ except for the facts: the former contains only those $P(\bar{a})$ with $P(\bar{a}) \in DB$, such that one P appears in $\mathcal{Q}$, or P is connected to P' in the graph $\mathcal{G}(IC)$, with P' appearing in $\mathcal{Q}$.

Proposition 4. $\Pi(DB, IC, \mathcal{Q}){\downarrow}\mathcal{Q}$ and $\Pi(DB, IC, \mathcal{Q})$ retrieve the same cautious/brave answers to query $\mathcal{Q}$. $\square$

Example 14. (example 11 cont.) The dependency graph $\mathcal{G}(IC)$ has the set of nodes $\{S, Q, R, T, W\}$, and edges $= \{(S, Q), (Q, R), (T, W)\}$. Thus, $\Pi(DB, IC, \mathcal{Q}) \downarrow \mathcal{Q}$ for $\mathcal{Q}: \ Ans(x) \leftarrow S\_(x, \mathbf{t}^{\star\star})$ contains as facts only those in relations $\{S, Q, R\}$, in this case, tuple $S(a)$. $\square$

Apart from the reduction of database facts to be involved in the computation of the stable models, it worth noticing that a system like DLV may bring into main memory the answers to a given query. In particular, this query could ask for the facts that are relevant to a second query as determined by dependency graph, so that they can be used for the computation of answers to the latter.

The selection of database facts performed here differs from the one described in [15], where database facts participating in violations of ICs are extracted from the database, and by doing so, the database is split into the "affected database" and the "safe database". This makes it possible to speed up the computation of database repairs, which are computed only for the affected part, and are next combined with the safe part.

5 Specification of Scalar Aggregation In Repair Programs

In [3] the notion of consistent answers for scalar aggregation queries in inconsistent database wrt functional dependencies (FDs) was defined. Aggregation queries are of the form: `SELECT f(...) FROM R`, where `f` is one of the aggregate operators *min, max, count, sum, avg*, which are applied over an attribute or the entire relation `R`. These queries return single numerical values. A consistent answer for a scalar aggregate query [3] is a shortest numerical interval $[a, b]$, where a, b are called the *greatest lower bound answer* (glb) and the *least upper bound answer* (lub), resp. This answer guarantees that the value of the scalar function evaluated in every repair can be found within the most informative interval.

Example 15. $DB = \{Emp(smith, 5000), Emp(smith, 8000), Emp(jones, 3000)\}$, violates $FD : name \rightarrow salary$, because $smith$ has two salaries. There are two repairs: $DB_1 = \{Emp(smith, 5000), Emp(jones, 3000)\}$ and $DB_2 = \{Emp(smith, 8000), Emp(jones, 3000)\}$. The consistent answer to query: `SELECT MAX(salary) FROM Emp` is the interval $[5000, 8000]$, 5000 is the *glb*, and 8000 is the *lub*. □

In [3] repairs are represented as independent sets in conflict graphs for FDs, which is a compact representation of all repairs. Repairs wrt FDs can also be specified by disjunctive logic programs, together with the aggregate query. The semantics of aggregation under stable model semantics for disjunctive program is investigated in [13, 17]. The implementation of aggregate queries in DLV is described in [14]. DLV currently implements *min, max, count, times, sum*, but not *avg* [18].

The repair programs will now contain rules defining aggregate functions, in DLV notation, rules of the form $A(w) \leftarrow \#f\{\bar{x}' : P\_(\bar{x}, \mathbf{t}^{\star\star})\} = w$, where A is a new predicate that is not present elsewhere in the program, f is an aggregation function, which is applied over variable $\bar{x}' \subseteq \bar{x}$ of predicate P, and w is a variable that stores the value returned by f in each stable model [17].

Example 16. (example 15 cont.) Program $\Pi(DB, IC)$ has the following rules[6]:
$Emp(smith, 5000).\ \ Emp(smith, 8000).\ \ Emp(jones, 3000).$
$Emp\_(x, y, \mathbf{f_a}) \vee Emp\_(x, z, \mathbf{f_a}) \leftarrow Emp\_(x, y, \mathbf{t}^\star), Emp\_(x, z, \mathbf{t}^\star), y \neq z, y \neq null,$
$z \neq null.$
$Emp\_(x, y, \mathbf{t}^\star) \leftarrow Emp\_(x, y, \mathbf{t_a}).$
$Emp\_(x, y, \mathbf{t}^\star) \leftarrow Emp(x, y).$
$Emp\_(x, y, \mathbf{t}^{\star\star}) \leftarrow Emp\_(x, y, \mathbf{t}^\star), not\ Emp\_(x, y, \mathbf{f_a}).$
$A(w) \leftarrow \#max\{y : Emp\_(x, y, \mathbf{t}^{\star\star})\} = w.$ □

Even though, the aggregate rule in the previous example satisfies the formalisms given in [17], the program will not run in DLV system. The reason is that DLV currently does not implement aggregations over predicates that are defined by unstratified or disjunctive rules, as in the previous case. In these cases there exists problems during the grounding process of DLV, which is performed before the computation of the stable models. For instance, variable w in the rule $A(w) \leftarrow \#max\{y : Emp\_(x, y, \mathbf{t}^{\star\star})\} = w$ is unbound during grounding, and DLV would have to compute all possible values for binding it. For functions max, min there are among the values taken by variable y, but for other functions as sum grounding could get very difficult. As a consequence, in order for DLV to answer queries involving functions *max* and *min*, rules have to be

[6] By corollary 1 this repair program does not have program denial constraints.

modified by inserting an extra argument that binds the aggregation variable. For instance, aggregation rule $A(w) \leftarrow \#max\{y : Emp\_(x, y, \mathbf{t}^{\star\star})\} = w$ is transformed into $A(w) \leftarrow \#max\{y : Emp\_(x, y, \mathbf{t}^{\star\star})\} = w, Emp\_(\_, w, \mathbf{t}^{\star\star})$, where variable w is bounded by atom $Emp\_(\_, w, \mathbf{t}^{\star\star})$.

Example 17. (example 16 cont.) The program $\Pi(DB, IC)$ with the aggregation rule $A(w) \leftarrow \#max\{y : Emp\_(x, y, \mathbf{t}^{\star\star})\} = w, Emp\_(\_, w, \mathbf{t}^{\star\star})$ now has the following stable models:

$$\begin{aligned}\mathcal{M}_1 = \{&Emp(smith, 5000), Emp(smith, 8000), Emp(jones, 3000), Emp\_(smith, 5000, \mathbf{t}^{\star}),\\ &Emp\_(smith, 8000, \mathbf{t}^{\star}), Emp\_(jones, 3000, \mathbf{t}^{\star}), Emp\_(smith, 5000, \mathbf{t}^{\star\star}),\\ &Emp\_(smith, 8000, \mathbf{f_a}), Emp\_(jones, 3000, \mathbf{t}^{\star\star}), \underline{A(5000)}\}\\ \mathcal{M}_2 = \{&Emp(smith, 5000), Emp(smith, 8000), Emp(jones, 3000), Emp\_(smith, 5000, \mathbf{t}^{\star}),\\ &Emp\_(smith, 8000, \mathbf{t}^{\star}), Emp\_(jones, 3000, \mathbf{t}^{\star}), Emp\_(smith, 5000, \mathbf{f_a}),\\ &Emp\_(smith, 8000, \mathbf{t}^{\star\star}), Emp\_(jones, 3000, \mathbf{t}^{\star\star}), \underline{A(8000)}\}\end{aligned}$$

The aggregation function returns 5000 as the maximum salary in the first repair, and 8000 in the second one.

In order to obtain consistent answers to aggregate queries, one can capture all the values returned by the aggregation function across the models, which can be achieved by posing the query $Ans(x) \leftarrow A(x)$ to the program $\Pi(DB, IC)$, evaluating it under the brave semantics. Here we obtain the values $Ans(5000), Ans(8000)$, which can be sorted to obtain the *glb* and *lub*. □

Notice that for *sum* and *count* it is not possible in general to bind variable w to a value in a database predicate.

6 Conclusions

In this paper, repair programs have been simplified and optimized by eliminating redundant rules, facts and annotations. In addition, important classes of ICs are identified for which repair programs can be specified without program denial constraints. The elimination of these rules becomes very important when magic sets techniques are applied with the DLV system. Magic sets techniques allows to focalize on part of repair programs and facts that are relevant to answer a query. It was shown that the magic set method is sound and complete when it is applied in disjunctive repair programs with program denial constraints.

In order to apply magic sets to repair programs in DLV, a suitable processing of program constraints (if the program contains them) has to be performed. This is due to the fact that currently DLV does not support magic sets for programs with denial constraints. In addition, DLV applies magic sets internally, without returning the rewritten program. This implies that adding later the program constraints to the rewritten program cannot be done by the user.

Moreover, the evaluation of programs in DLV was also improved by involving only relevant facts in the computation of query answers, so that now only a smaller portion of the database is imported into DLV.

We explore the aggregation capabilities of DLV system for computing consistent answers to scalar aggregation queries as defined in [3]. The current version of DLV implements five aggregation functions, but with some restrictions that are satisfied by our repair programs only for functions *max* and *min*.

We are currently developing a system for computing consistent query answers based repair programs. Currently, the system implements the logic approach presented in [7], with some of the structural optimizations presented in section 3.

In addition, we are working in the identification of classes of ICs and queries for which, the *well-founded semantics* of logic programs [22] can be used, instead of the stable models semantics. Preliminary research in this direction can be found in [2], for slightly different specification programs. This could be interesting due to the fact that the well-founded semantics has lower computational complexity than the stable model semantics [12], and efficient implementations are available [31].

It would be also interesting to extend the techniques described in [15] for splitting the database into the affected and safe parts to referential integrity constraints under the semantics of their satisfaction in presence of null values.

Extensions of consistent query answering to aggregate queries with `GROUP BY` using logic programs would be also interesting.

Acknowledgements: Research supported by NSERC (Grant 250279-02) and the University of Bio-Bio (Chile). L. Bertossi is Faculty Fellow of IBM CAS (Toronto). We are grateful to Nicola Leone, Gianluigi Greco and Wolfgang Faber for their support and information in relation to the DLV system.

References

[1] Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. In *Proc. 18th ACM Symposium on Principles of Database Systems (PODS 99)*, ACM Press, 1999, pp. 68-79.

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

[3] Arenas, M., Bertossi, L., Chomicki, J., He, X., Raghavan, V., and Spinrad, J. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 2003, 296:405–434.

[4] Barcelo, P. and Bertossi, L. Logic Programs for Querying Inconsistent Databases. In *Proc. 5th International Symposium on Practical Aspects of Declarative Languages (PADL 03)*. Springer LNCS 2562, 2003, pp. 208–222.

[5] Barcelo, P., Bertossi, L. and Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. Chapter in book *Semantics of Databases*, Springer LNCS 2582, 2003, pp. 1–27.

[6] Beeri, C. and Ramakrishnan, R. On the Power of Magic. In *Proc. 6th ACM Symposium on Principles of Database Systems (PODS 87)*, ACM Press, 1987, pp. 269-284.

[7] Bravo, L. and Bertossi, L. Consistent Query Answering under Inclusion Dependencies. 14th Annual IBM Centers for Advanced Studies Conference (CASCON 2004), pp. 202-216.

[8] Cali, A., Lembo, D. and Rosati, R. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *Proc. Symposium on Principles of Database Systems (PODS 03)*, ACM Press, 2003, pp. 260-271.

[9] Ceri, S., Gottlob, G. and Tanca, L. *Logic Programming and Databases*. Springer-Verlag, 1990.

[10] Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance Using Tuple Deletions. *Information and Computation*, 2005, 197(1-2):90-121.

[11] Cumbo, C., Faber, W., Greco, G. and Leone, N. Enhancing the Magic-Set Method for Disjunctive Datalog Programs. In *Proc. of the 20th International Conference on Logic Programming (ICLP 04)*, Springer LNCS 3132, 2004, pp. 371–385.

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

[13] Dell'Armi, T., Faber, W., Ielpa, G., Leone, N. and Pfeifer, G. Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI 03)*, 2003, Morgan Kaufmann, pp. 847-852.

[14] Dell'Armi, T., Faber, W., Ielpa, G., Leone, N. and Pfeifer, G. Aggregate Functions in DLV. In *Proc. Answer Set Programming: Advances in Theory and Implementation*, 2003, Marina de Vos and Alessandro Provetti, pp. 274–288.

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

[16] Faber, W., Greco, G. and Leone, N. Magic Sets and their Application to Data Integration. In *Proc. International Conference on Database Theory (ICDT 05)*, Springer LNCS 3363, 2005, pp. 306-320.

[17] Faber, W., Leone, N. and Pfeifer, G. Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. In *Proc. 9th European Conference on Artificial Intelligence (JELIA 2004)*, Springer LNCS 3229, 2004, pp. 200–212.

[18] Faber, W. Unfounded Sets for Disjunctive Logic Programs with Arbitrary Aggregates. In *Logic Programming and Nonmonotonic Reasoning, 8th International Conference (LP-NMR'05)*, 2005, Springer Verlag, To appear.

[19] Fuxman, A. and Miller, R. First-Order Query Rewriting for Inconsistent Databases. In *Proc. International Conference on Database Theory (ICDT 05)*, Springer LNCS 3363, 2004, pp. 337-354.

[20] Gelfond, M. and Lifschitz, V. The Stable Model Semantics for Logic Programming. In *Logic Programming, Proceedings of the 5th International Conference and Symposium (ICLP/SLP 88)*, MIT Press, 1988, pp. 1070-1080.

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

[22] Van Gelder, A., Ross, K.A., Schlipf, J.S. Unfounded Sets and Well-Founded Semantics for General Logic Programs. In *Proc. Symposium on Principles of Database Systems (PODS 88)*, ACM Press, 1988, pp. 221-230.

[23] Greco, S. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. In *IEEE Transactions on Knowledge and Data Engineering*, 2003, 15(2):368-385.

[24] Greco, G., Greco, S., Trubtsyna, I. and Zumpano, E. Optimization of Bound Disjunctive Queries with Constraints. To Appear in Theory and Practice of Logic Programming, (oai:arXiv.org:cs/0406013), 2004.

[25] Lenzerini, M. Data Integration: A Theoretical Perspective. In *Proc. ACM Symposium on Principles of Database Systems (PODS 02)*, ACM Press, 2002, pp. 233-246.

[26] Lifschitz, V. and Turner, H. Splitting a Logic Program. In *Proc. ICLP 94*, pp. 23–37, MIT Press, 1994.

[27] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. The DLV System for Konwledge Representation and Reasoning. To appear in *ACM Transactions on Computational Logic*, arXiv.org paper cs.LO/0211004.

[28] Lloyd, J.W. *Foundations of Logic Programming*. Second ed., Springer-Verlag, 1987.

[29] Przymusinski, T.C. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9(3/4):401–424, 1991.

[30] Ross, K. Modular Stratification and Magic Sets for Datalog Programs with Negation. *J. ACM*, 1994, 41(6), pp. 1216-1266.

[31] Sagonas, K.F., Swift, T. and Warren, D.S. XSB as an Efficient Deductive Database Engine. In *Proc. International Conference on Management of Data (SIGMOD 94)*, ACM Press, 1994, pp. 442-453.

7 Appendix: Proofs

Before proving the propositions and theorems, we introduce some definitions that will be used in them.

Definition 8. [11] Given a set S of ground rules of a program P ($ground(P)$), $R(S)$ denotes the set $\{r \in ground(P) | \exists\, r' \in S, \exists\, q \in B(r') \cup H(r')$ such that $q \in H(r)\}$. $B(r')$ and $H(r')$ stand for the body of rule r', and the head of r' respectively. Then, $rel(\mathcal{Q}, P)$ is the least fixed point of the following succession $rel_0(\mathcal{Q}, P) = \{r \in ground(P) | \exists\, ground(q) \in \mathcal{Q} \cap H(r)\}$, and $rel_{i+1}(\mathcal{Q}, P) = R(rel_i(\mathcal{Q}, P))$, for each $i > 0$.

Given a model M and a predicate symbol g, define $M[g]$ as the set of atoms in M whose predicate symbol is g. In addition, given a program P, $P[g]$ is the set of rules of P whose head contains symbol g.

Then, given a model M and a program P, $M[P]$ is the set of atoms in M whose predicate symbol appears in the head of some rule in program P. And, given a set of interpretations S, $S[g] = \{M[g] | M \in S\}$, therefore $S[P] = \{M[P] | M \in S\}$ [11]. $\Box$

Proof of Proposition 1: Consider Definition 4 of a locally stratified program. Given a database DB and a set of RIC-acyclic universal and referential ICs, let $\{V_1, \ldots, V_r\}$ be the set of vertices of $\mathcal{G}^c(IC)$. Since this graph is obtained by contracting the vertices of $\mathcal{G}(IC)$, each vertex in $\mathcal{G}^c(IC)$ is a set of predicates of $\mathcal{R}$. In fact, $\bigcup_{i=1}^{r} V_i = \mathcal{R}$ and $V_j \cap V_k = \emptyset$ for $V_j, V_k \in V$. Since $\mathcal{G}^c(IC)$ is acyclic, we can safely assume that the vertices are numbered in a topological ordering, i.e for every edge (V_i, V_j), we have $i < j$. Then, for $\Pi(DB, IC)$ and $\mathcal{U}^\star = (\mathcal{U} \cup null)$ we can consider the following *strata*:

$S_0 = \{p(\bar{x}, \mathbf{t_d}) \mid p \in \mathcal{R}$ and $\bar{x} \in \mathcal{U}^\star\} \cup \{dom(x) \mid x \in \mathcal{U}^\star\}$

$S_1 = \{p(\bar{x}, y) \mid p \in \mathcal{V}_r, \bar{x} \in \mathcal{U}^\star$ and $y \in \{\mathbf{t}^\star, \mathbf{t_a}, \mathbf{f_a}\}\}$

$S_2 = \{aux_i(\bar{x}) \mid V_r$ has an incoming edge in $\mathcal{G}^c(IC)$ corresponding to the referential integrity constraint IC_i and $\bar{x} \in \mathcal{U}^\star\}$

$S_3 = \{p(\bar{x}, y) \mid p \in \mathcal{V}_{r-1}, \bar{x} \in \mathcal{U}^\star$ and $y \in \{\mathbf{t}^\star, \mathbf{t_a}, \mathbf{f_a}\}\}$

$S_4 = \{aux_i(\bar{x}) \mid V_r$ has an incoming edge in $\mathcal{G}^c(IC)$ corresponding to the referential integrity constraint IC_i and $\bar{x} \in \mathcal{U}^\star\}$

...

S_i(for $i \leq (2r-1)$ and odd)$= \{p(\bar{x}, y) \mid p \in \mathcal{V}_{r-\lfloor \frac{i-1}{2} \rfloor}, \bar{x} \in \mathcal{U}^\star$ and $y \in \{\mathbf{t}^\star, \mathbf{t_a}, \mathbf{f_a}\}\}$

S_i(for $i \leq (2r-1)$ and even)$= \{aux_i(\bar{x}) \mid V_{r-\lfloor\frac{i-1}{2}\rfloor}$ has an incoming edge in $\mathcal{G}^c(IC)$ corresponding to the referential integrity constraint IC_i and $\bar{x} \in \mathcal{U}^\star\}$
$\ldots$
$S_{2r-1} = \{p(\bar{x}, y) \mid p \in \mathcal{V}_1, \bar{x} \in \mathcal{U}^\star$ and $y \in \{\mathbf{t}^\star, \mathbf{t_a}, \mathbf{f_a}\}\}$
$S_{2r} = \{p(\bar{x}, y) \mid p \in \mathcal{R}, \bar{x} \in \mathcal{U}^\star$ and $y \in \{\mathbf{t}^{\star\star}\}\}$
It is easy to check that this $strata$ satisfies the needed conditions on every rule of the program (without considering the program denial constraints) and therefore that the program is locally stratified. □

Proof of Proposition 2: By contradiction, suppose that there exists a stable model $\mathcal{M}$ for $\Pi'(DB,\, IC)$ having both $P\_(\bar{a}, \mathbf{t_a})$, and $P\_(\bar{a}, \mathbf{f_a})$ for a given predicate P. Then, since $\mathcal{M}$ is a model of $\Pi'(DB,\, IC)$, the program does not have a program denial constraint $\leftarrow P\_(\bar{a}, \mathbf{t_a}), P\_(\bar{a}, \mathbf{f_a})$. This implies that P is a sink (or source) node in $\mathcal{G}(IC)$. In addition, since $P\_(\bar{a}, \mathbf{t_a})$ and $P\_(\bar{a}, \mathbf{f_a})$ are in $\mathcal{M}$, program $\Pi'(DB,\, IC)$ has at least a rules defining $P\_(\bar{x}, \mathbf{t_a})$ and another defining $P\_(\bar{x}, \mathbf{f_a})$. However, if these rules are in program $\Pi'(DB,\, IC)$ then P cannot be a sink (or source) node in graph $\mathcal{G}(IC)$. We have reached a contradiction. □

Before giving the proof of theorem 1, we make the following simplifications:

- Π denotes the repair program $\Pi(DB, IC)$.
- $\Pi^\star$ denotes the repair program $\Pi^\star(DB, IC)$.

This is possible since the database instance DB, the set of ICs IC, and the query $\mathcal{Q}$ do not change in any of the proof.

In addition, we define two sets, AC contains the following annotation constants: $\{\mathbf{f_a}, \mathbf{t_a}, \mathbf{t_d}, \mathbf{t}^\star, \{\}\}$, where $\{\}$ is use to refer to the "no annotation" of database facts in program $\Pi^\star$. The second set is IR which is composed by constant $\mathbf{t}^{\star\star}$.

Then, $SM(\Pi)[AC]$ is the set of stable models of program Π restricted to the atoms that have annotation constants in AC (including database facts of program $\Pi^\star$). $\Pi[AC]$ is the repair program restricted to the rules whose head atom contains one of the annotation arguments in AC. And $\Pi[IR]$ denotes the interpretation rules of the ground repair program. Clearly $\Pi = \Pi[IR] \cup \Pi[AC]$.

Proof of Theorem 1: The proof is divided in the following items:

1. The elimination of $\mathbf{t_d}$ annotation of repair programs does not affect the semantics of the program.
   In Π database facts are atoms of the form $P(\bar{a}, \mathbf{t_d})$, where $\mathbf{t_d}$ is an extra argument given by the program to identify database facts from others atoms in the program. In program $\Pi^\star$ that annotation is eliminated and database facts are used as they come from the database, e.g. they are atoms of the form $P(\bar{a})$. It is easy to see that $P(\bar{a}, \mathbf{t_d})$ and $P(\bar{a})$ refer to the same database facts, given the fact that they are retrieved from the same database instance DB. So that, $P(\bar{a}, \mathbf{t_d})$ and $P(\bar{a})$ are equivalent.
   Due to the elimination of $\mathbf{t_d}$ annotation, in program $\Pi^\star$ the version of P that is expanded with others annotations is replaced by an underscored version, e.g.

$P(\bar{a}, \mathbf{t_a})$, becomes $P\_(\bar{a}, \mathbf{t_a})$, etc. This is just a syntactic change. So that, atom $P(\bar{a}, \mathbf{t_a})$ in program Π has the same meaning as atom $P\_(\bar{a}, \mathbf{t_a})$ in program $\Pi^\star$.
As a consequence, the elimination of $\mathbf{t_d}$ annotation does not alter the semantics of the repair program.

2. The replacement of $dom(\bar{x})$ in rules, by conditions of the form $\bar{x} \neq null$ does not affect the semantics of the program.
   Let us remember the semantic of constraint satisfaction specified by repair programs. First, UICs are satisfied if they hold for tuples with non-null values. Second, RICs are classically satisfied when universally quantified variables in the RIC take values different from null, and existentially quantified variables taken any value. It follows from the previous, that we check consistency of ICs for tuples whose values are different from the "null" constant. And that is precisely what we achieve by adding conditions of the form $\bar{x} \neq null$, for every variable in the IC rules of repair programs. So, it is easy to see that adding those conditions is equivalent, to restrict the values of variables to be part of the database domain which is achieved in Π with the predicate $dom(\bar{x})$.
   Those conditions are just needed in the IC rules, because they are the ones that check the satisfaction of ICs. Therefore, the elimination of $dom(\bar{x})$ atoms does not affect the semantics of the program.
3. The elimination of some denial constraints does not affect the semantics of the program.
   It follows from Proposition 2.
4. The interpretation rules in program $\Pi^\star$ and Π define the same set of atoms annotated with $\mathbf{t}^{\star\star}$.
   From the previous items we know that:

   Fact A.1. For every stable model M that belongs to $SM(\Pi)[AC]$, there exists a stable model M' that belongs to $SM(\Pi^\star)[AC]$, such that $M = M'$, wrt atoms with annotations constants: $\{\mathbf{f_a}, \mathbf{t_a}, \mathbf{t}^\star\}$.

   Fact A.2. For every stable model M' that belongs to $SM(\Pi^\star)[AC]$, there exists a stable model M that belongs to $SM(\Pi)[AC]$, such that $M' = M$, wrt atoms with annotations constants: $\{\mathbf{f_a}, \mathbf{t_a}, \mathbf{t}^\star\}$.

   It is easy to see that program Π ($\Pi^\star$) can be split into a bottom program $\Pi[AC]$ ($\Pi^\star[AC]$) and a top program $\Pi[IR]$ ($\Pi^\star[IR]$) using as a splitting set all the atoms except the ones annotated with $\mathbf{t}^{\star\star}$ [26]. This implies that the programs can be hierarchically evaluated in the following way: the models of program Π are $SM(\Pi) = \bigcup_M SM(M \cup \Pi[IR])$, for each stable model M in $SM(\Pi)[AC]$.
   So, we prove that:
   – For every stable model M" that belongs to $SM(M \cup \Pi[IR])$ with M in $SM(\Pi)[AC]$, there exists a stable model $M^\star$ that belongs to $SM(M' \cup \Pi^\star[IR])$, with M' in $SM(\Pi^\star)[AC]$, such that M"$[\mathbf{t}^{\star\star}] = M^\star[\mathbf{t}^{\star\star}]$.
     By contradiction, let us assume that there exists a stable model M" in $SM(M \cup \Pi[IR])$ with M in $SM(\Pi)[AC]$, and there is not a stable model $M^\star$ that belongs to $SM(M' \cup \Pi^\star[IR])$ with M' in $SM(\Pi^\star)[AC]$, such that M"$[\mathbf{t}^{\star\star}] = M^\star[\mathbf{t}^{\star\star}]$
     We have two cases depending if the repair obtained from M" is empty or not.

- First, the repair obtained from M" is empty. So M" does not have atoms with the $\mathbf{t}^{\star\star}$ constant. In this case $M" = M$ with M in $SM(\Pi)$ $[AC]$. Then, according to Fact 1 we know that there exists a model M' in $SM(\Pi^\star)$ $[AC]$ such that $M = M'$. So given the fact that $M" = M$, we now also have that $M" = M'$. Now, if $M^\star$ has no atoms with $\mathbf{t}^{\star\star}$, $M"[\mathbf{t}^{\star\star}]$ would be equal to $M^\star[\mathbf{t}^{\star\star}]$ (both would be empty) and this would lead to a contradiction. Then, $M^\star[\mathbf{t}^{\star\star}]$ should not be empty. Then, there exists an atom $P\_(\bar{c}, \mathbf{t}^{\star\star})$ in $M^\star$. Then, M' has $P\_(\bar{c}, \mathbf{t}^\star)$ and does not have $P\_(\bar{c}, \mathbf{f_a})$. If M' has $P\_(\bar{c}, \mathbf{t}^\star)$ then $P(\bar{c})$ is true or $P\_(\bar{c}, \mathbf{t_a})$ is true in M'. However, if either of both situations happens, and given the facts that $P\_(\bar{c}, \mathbf{f_a})$ is false in M', and $M = M'$, then M" satisfies $P(\bar{c}, \mathbf{t}^{\star\star})$ as well. That because of the interpretation rules in $\Pi[IR]$. But, $M"[\mathbf{t}^{\star\star}] = \emptyset$. We have reached a contradiction.
- Second, the repair obtained from M" is not empty. In this case, there exists a tuple $P(\bar{c}, \mathbf{t}^{\star\star})$ in M" such that there is no model $M^\star$ that satisfies $P\_(\bar{c}, \mathbf{t}^{\star\star})$. If atom $P(\bar{c}, \mathbf{t}^{\star\star})$ is true in M" then we have that $P(\bar{c}, \mathbf{t_d})$ is in M, (M in $SM(\Pi)[AC]$) in which case $P(\bar{c}, \mathbf{f_a})$ is not in M, or $P(\bar{c}, \mathbf{t_d})$ is not in M, in which case $P(\bar{c}, \mathbf{t_a})$ is in M. In both cases we have that atom $P(\bar{c}, \mathbf{t}^\star)$ is true in M. In addition, because of Fact 1 we know that exists a stable model M' in $SM(\Pi^\star)[AC]$, such that $M = M'$. Now, there are two cases to consider: $P(\bar{c}, \mathbf{t_d})$ is in M and $P(\bar{c}, \mathbf{t_d})$ is not in M.
First, for $P(\bar{c}, \mathbf{t_d})$ in M, we have that since $P(\bar{c}, \mathbf{t}^\star)$ is in M, $P(\bar{c}, \mathbf{f_a})$ has to be false in M. Then, since $M = M'$ and because of the interpretation rule of $\Pi^\star$: $P\_(\bar{c}, \mathbf{t}^{\star\star}) \leftarrow P\_(\bar{c}, \mathbf{t}^\star),\ not\ P\_(\bar{c}, \mathbf{f_a})$, we have that there exists a model $M^\star$ such that $P\_(\bar{c}, \mathbf{t}^{\star\star})$ is in $M^\star$. Then we have reached a contradiction. Second, for $P(\bar{c}, \mathbf{t_d})$ not in M, we have that since $P(\bar{c}, \mathbf{t}^\star)$ is in M, $P(\bar{c}, \mathbf{t_a})$ has to be true in M and $P(\bar{c}, \mathbf{f_a})$ has to be false in M. Then, since $M = M'$ and because of the interpretation rule of $\Pi^\star$: $P\_(\bar{c}, \mathbf{t}^{\star\star}) \leftarrow P\_(\bar{c}, \mathbf{t}^\star),\ not\ P\_(\bar{c}, \mathbf{f_a})$, we have that there exists a model $M^\star$ such that $P\_(\bar{c}, \mathbf{t}^{\star\star})$ is in $M^\star$. We have reached a contradiction.

– For every stable model $M^\star$ that belongs to $SM(M' \cup \Pi^\star[IR])$, with M' in $SM(\Pi^\star)[AC]$, there exists a stable model M" that belongs to $SM(M \cup (\Pi)[IR])$ with M in $SM(\Pi)[AC]$, such that $M^\star[\mathbf{t}^{\star\star}] = M"[\mathbf{t}^{\star\star}]$.
By contradiction, let us assume that there exists a model $M^\star$ that belongs to $SM(M' \cup \Pi^\star[IR])$ with M' in $SM(\Pi^\star)[AC]$, and there is not a stable model M" that belongs to $SM(M \cup \Pi[IR])$ with M in $SM(\Pi)[AC]$, such that $M^\star[\mathbf{t}^{\star\star}] = M"[\mathbf{t}^{\star\star}]$. Here we have two cases depending if the repair obtained from $M^\star$ is empty or not.
- First, the repair obtained from $M^\star$ is empty. So $M^\star$ does not have atoms with the $\mathbf{t}^{\star\star}$ constant. In this case $M^\star = M'$ with M' in $SM(\Pi^\star)$ $[AC]$. Then, because of Fact 2 we know that there exists a model M in $SM(\Pi)$ $[AC]$ such that $M' = M$. So given the fact that $M^\star = M'$, we now also have that $M^\star = M$. Now, if $M^\star$ has no atoms with $\mathbf{t}^{\star\star}$, $M^\star[\mathbf{t}^{\star\star}]$ would be equal to $M"[\mathbf{t}^{\star\star}]$ (both would be empty) and this would lead to a contradiction. Then, $M"[\mathbf{t}^{\star\star}]$ should not be empty.

Then, there exists a tuple $P(\bar{c}, \mathbf{t}^{\star\star})$ in M". Now, there are two cases to analyze, $P(\bar{c}, \mathbf{t_d})$ is in M or $P(\bar{c}, \mathbf{t_d})$ is not in M. First, if $P(\bar{c}, \mathbf{t_d})$ is in M, then, $P(\bar{c})$ is in M' and $P\_(\bar{c}, \mathbf{t}^{\star})$ is in M'. Also, since $P(\bar{c}, \mathbf{t}^{\star\star})$ is in M", $P(\bar{c}, \mathbf{f_a})$ is not in M and $P(\bar{c}, \mathbf{f_a})$ is not in M'. Given the interpretation rule in $\Pi^{\star}$: $P(\bar{c}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{c}, \mathbf{t_d}),\ not\ P(\bar{c}, \mathbf{f_a})$, we have $P\_(\bar{c}, \mathbf{t}^{\star\star})$ is in $M^{\star}$. But, $M^{\star}[\mathbf{t}^{\star\star}]$ is empty. We have reached a contradiction. Now, we need to analyze for $P(\bar{c}, \mathbf{t_d})$ is not in M. Since $P(\bar{c}, \mathbf{t}^{\star\star})$ is in M", $P(\bar{c}, \mathbf{f_a})$ is not in M and $P(\bar{c}, \mathbf{t_a})$ is in M. Then, given the fact that $M = M'$, then $P\_(\bar{c}, \mathbf{t_a})$ is in M', and $P\_(\bar{c}, \mathbf{t}^{\star})$ is in M'. So given the interpretation rule in $\Pi^{\star}$: $P(\bar{c}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{c}, \mathbf{t_d}),\ not\ P(\bar{c}, \mathbf{f_a})$, we have $P\_(\bar{c}, \mathbf{t}^{\star\star})$ is in $M^{\star}$. But, $M^{\star}[\mathbf{t}^{\star\star}]$ is empty. We have reached a contradiction.

- Second, the repair obtained from $M^{\star}$ is not empty, so there exists $P\_(\bar{c}, \mathbf{t}^{\star\star})$ in $M^{\star}$. If atom $P\_(\bar{c}, \mathbf{t}^{\star\star})$ is true in $M^{\star}$ then we have that $P\_(\bar{c}, \mathbf{t}^{\star})$ is true in M' (M' is in $SM(\Pi^{\star})[AC]$), and $P\_(\bar{c}, \mathbf{f_a})$ is false in M'. If $P\_(\bar{c}, \mathbf{t}^{\star})$ is true, then either $P(\bar{c})$ or $P\_(\bar{c}, \mathbf{t_a})$ are true in M'. In addition, because of Fact 2 we know that exists a model M in $SM(\Pi)[AC]$, such that $M' = M$. Now there are two cases to consider: $P(\bar{c})$ is in M' or $P(\bar{c})$ is not in M'. First we will assume that $P(\bar{c})$ is true in M', and therefore that $P\_(\bar{c}, \mathbf{f_a})$ is false. Then because $M = M'$, and by using the interpretation rule: $P(\bar{c}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{c}, \mathbf{t_d}),\ not\ P(\bar{c}, \mathbf{f_a})$ of Π, $P(\bar{c}, \mathbf{t}^{\star\star})$ is in M". Then M" exists and we have reached a contradiction. Now, if $P(\bar{c})$ is not in M' we have that $P\_(\bar{c}, \mathbf{t_a})$ is in M'. Then because $M = M'$, and by using the interpretation rule: $P(\bar{c}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{c}, \mathbf{t_a})$ of Π, $P(\bar{c}, \mathbf{t}^{\star\star})$ is in M". Then M" exists and we have reached a contradiction.

$\square$

Before proving Theorem 2 we need to introduce some propositions and lemmas.

By personal communication we know that the following lemmas, which are a combination of results presented in [11] and [16] have been proved, but they are not published yet.

Lemma A.1. [11, 16] Given a datalog program P with unstratified negation, where negation is involved only in even cycles, for every stable model M' that belongs to $SM(\mathcal{MS}(P, \mathcal{Q}))$, there exists a stable model M that belongs to $SM(rel(\mathcal{Q}, P))$ such that $M = M'[rel(\mathcal{Q}, P)]$, $\square$

Lemma A.2. [11, 16] Given a datalog program P with unstratified negation, where negation is involved only in even cycles, for every stable model M that belongs to $SM(rel(\mathcal{Q}, P))$, there exists a stable model M' that belongs to $SM(\mathcal{MS}\ (P, \mathcal{Q}))$ such that $M = M'[rel(\mathcal{Q}, P)]$. $\square$

Note that the correspondence between the stable models (SM) of the magic rewritten program $\mathcal{MS}(P, \mathcal{Q})$, and the SM of program $rel(\mathcal{Q}, \Pi)$ is established by focusing on non-magic atoms only. Which is achieved in Lemma A.1 by condition $M = M'[rel(\mathcal{Q}, P)]$, and in Lemma A.2 by condition $M = M'[rel(\mathcal{Q}, P)]$.

Here, we use Π to refer to $\Pi(DB, IC, \mathcal{Q})$; Π^- to refer to $\Pi^-(DB, IC, \mathcal{Q})$. $\mathcal{MS}(\Pi^-)$ refers to the MS method for disjunctive programs without denial constraints, and it is applied as presented in [11, 16]. We use $\mathcal{MS}^{\leftarrow}(\Pi)$ to refer to the MS method presented in section 4, for disjunctive programs with denial constraints. Then, $\mathcal{MS}^{\leftarrow} = \mathcal{MS}(\Pi^-) + PD$. In addition, $PD_{\mathcal{Q}}$ is the set of program denial constraints involving predicates that are connected to the query predicates in the dependency graph $\mathcal{G}(IC)$. $PD^{\star}_{\mathcal{Q}}$ is a program containing rules of the form $Ans(x) \leftarrow \underline{P}(x, \mathbf{t_a}), \underline{P}(x, \mathbf{f_a})$, for each program denial constraint d in $PD_{\mathcal{Q}}$. The repair program Π is coherent if it satisfies the set of program constraints PD, and the program $rel(\mathcal{Q}, \Pi)$ is coherent if it satisfies the set of program constraints $PD_{\mathcal{Q}}$.

Proposition A.1. For a disjunctive program Π, and the set of program denial constraints $PD_{\mathcal{Q}}$, we have that: $rel(PD^{\star}_{\mathcal{Q}} \cup \mathcal{Q}, \Pi^-) \equiv rel(\mathcal{Q}, \Pi^-)$. □

Proof. It is easy to see that $rel(PD^{\star}_{\mathcal{Q}} \cup \mathcal{Q}, \Pi^-)$ is equivalent to: $rel(PD^{\star}_{\mathcal{Q}}, \Pi^-) \cup rel(\mathcal{Q}, \Pi^-)$. So it is sufficient to prove that program $rel(PD^{\star}_{\mathcal{Q}}, \Pi^-) \subseteq rel(\mathcal{Q}, \Pi^-)$.

First, if there are not program denial constraints relevant to answer the query, then program $PD^{\star}_{\mathcal{Q}}$ has no rules, and $rel(PD^{\star}_{\mathcal{Q}}, \Pi^-)$ is empty, and we have that $rel(PD^{\star}_{\mathcal{Q}} \cup \mathcal{Q}, \Pi^-) \equiv rel(\mathcal{Q}, \Pi^-)$ trivially holds. So, we focalize in the case where there are program denial constraints relevant to answer the query.

For simplicity let us assume query $\mathcal{Q} : Ans(x) \leftarrow \underline{s}(\bar{a}, \mathbf{t}^{\star\star})$, and there exists one program denial constraint associated with the query, such that, $PD^{\star}_{\mathcal{Q}}$ is a rule of the form: $ans(x) \leftarrow \underline{s}(\bar{a}, \mathbf{t_a}), \underline{s}(\bar{a}, \mathbf{f_a})$. It is easy to see that: $rel_0(PD^{\star}_{\mathcal{Q}}, \Pi^-)$ is composed by the rules of program Π^- whose heads contains either atom $\underline{s}(\bar{a}, \mathbf{t_a})$ or $\underline{s}(\bar{a}, \mathbf{f_a})$.

In the other hand, $rel_0(\mathcal{Q}, \Pi^-)$ is composed by the interpretation rule: $\underline{s}(\bar{a}, \mathbf{t}^{\star\star}) \leftarrow \underline{s}(\bar{a}, \mathbf{t}^{\star}),\ not\ \underline{s}(\bar{a}, \mathbf{f_a})$. Then, $rel_1(\mathcal{Q}, \Pi^-)$ is: $rel_0(\mathcal{Q}, \Pi^-)$ plus rules of Π^- whose head atom is $\underline{s}(\bar{a}, \mathbf{t}^{\star})$ or $\underline{s}(\bar{a}, \mathbf{f_a})$. Then, $rel_2(\mathcal{Q}, \Pi^-)$ is: $rel_1(\mathcal{Q}, \Pi^-)$ plus rules of Π^- whose head atom is $\underline{s}(\bar{a}, \mathbf{t_a})$ or database facts of the form $\underline{s}(\bar{a})$. So, we have that $rel_0(PD^{\star}_{\mathcal{Q}}, \Pi^-) \subseteq rel_2(\mathcal{Q}, \Pi^-)$, and as a consequence $rel(PD^{\star}_{\mathcal{Q}}, \Pi^-) \subseteq rel(\mathcal{Q}, \Pi^-)$ holds. □

Since our disjunctive repair programs are locally stratified (Proposition 1, and given the fact that MS applied over this kind of program without denial constraints, i.e. $\mathcal{MS}(\Pi^-)$ is sound a complete, we will use Lemmas A.1 and A.2 in the proof of the following propositions.

Proposition A.2. For every stable model M' that belongs to $SM(\mathcal{MS}^{\leftarrow}(\Pi))$, there exists a stable model M that belongs to $SM(rel(\mathcal{Q}, \Pi^-))$ such that M is coherent, i.e. it satisfies the set of program constraints $PD_{\mathcal{Q}}$, and $M = M'[rel(\mathcal{Q}, \Pi^-)]$. □

Proof. By contradiction, let us assume that there exists a stable model M' in $SM(\mathcal{MS}^{\leftarrow}(\Pi))$ such that there is not a stable model M in $SM(rel(\mathcal{Q}, \Pi^-))$, where M is coherent, and $M = M'[rel(\mathcal{Q}, \Pi^-)]$.

M' is in $SM(\mathcal{MS}^{\leftarrow})$, then since $\mathcal{MS}^{\leftarrow} = \mathcal{MS}(\Pi^-) + PD$, M' is in $SM(\mathcal{MS}(\Pi^-))$. Now, by Lemma A.1, there exists a model M" in $SM(rel(\mathcal{Q}, \Pi^-))$ such that M" $= M'[rel(\mathcal{Q}, \Pi^-)]$. Because of the hypothesis M" is incoherent. We have two cases:

– M” is incoherent wrt a denial d that is in $PD \backslash PD_{\mathcal{Q}}$. We know that M” is a model of $rel(\mathcal{Q}, \Pi^-)$. By Proposition A.1 M” is a model of $rel(PD^{\star}_{\mathcal{Q}} \cup \mathcal{Q}, \Pi^-)$. Since d is in $PD \backslash PD_{\mathcal{Q}}$, there is no rule in $rel(PD^{\star}_{\mathcal{Q}} \cup \mathcal{Q}, \Pi^-)$ defining atoms that are relevant to it. Then, M” has no atoms relevant to it and it cannot be violated. We have reached a contradiction.
– M” is incoherent wrt a denial in d that is in $PD_{\mathcal{Q}}$, e.g. $d :\leftarrow \underline{S}(x, \mathbf{t_a}), \underline{S}(x, \mathbf{f_a})$. We have that $\underline{S}(x, \mathbf{t_a}), \underline{S}(x, \mathbf{f_a})$ are in M” and that $M” = M'[rel(\mathcal{Q}, \Pi^-)]$, therefore $\underline{S}(x, \mathbf{t_a})$, $\underline{S}(x, \mathbf{f_a})$ are in M'. But M' satisfies $PD_{\mathcal{Q}}$. We have reached a contradiction. □

Proposition A.3. For every stable model M that belongs to $SM(rel(\mathcal{Q}, \Pi^-))$, such that M is coherent, i.e. it satisfies the set of program constraints $PD_{\mathcal{Q}}$, there exists a stable model M' that belongs to SM $(\mathcal{MS}^{\leftarrow}(\Pi))$, such that $M = M'[rel(\mathcal{Q}, \Pi^-)]$. □

Proof. M is in $SM(rel(\mathcal{Q}, \Pi^-))$ then by lemma A.2, there exists a stable model M” in SM $(\mathcal{MS}(\Pi^-))$, such that $M = M”[rel(\mathcal{Q}, \Pi^-)]$. M is coherent, therefore M” is coherent as well. Then, since M” is coherent wrt $PD_{\mathcal{Q}}$, it will also be a model of $\mathcal{MS}(\Pi^-) + PD_{\mathcal{Q}}$. Now, since $\mathcal{MS}(\Pi^-)$ does not have rules for predicates in $PD \smallsetminus PD_{\mathcal{Q}}$, then M” is also a model of $\mathcal{MS}^{\leftarrow}(\Pi) = \mathcal{MS}(\Pi^-) + PD$. □

Proof of Theorem 2: In order to prove the soundness and completeness of MS applied over disjunctive programs with program denial constraints, and having propositions A.1, A.2 and A.3 we just need to prove that: $rel(\mathcal{Q}, \Pi) \cup PD_{\mathcal{Q}} \equiv_{\mathcal{Q}} \Pi$, where $\Pi = \Pi(DB, IC, \mathcal{Q})$, under both cautious and brave semantics. Note that we need to add $PD_{\mathcal{Q}}$ to $rel(\mathcal{Q}, \Pi)$ since in the previous propositions we are referring to coherent models only.

It is easy to see that the ground program Π can be split into a bottom program $\Pi_b = (rel(\mathcal{Q}, \Pi) \cup PD_{\mathcal{Q}})$ and a top program $\Pi_t = (\Pi \setminus \Pi_b)$ [26]. Using as a splitting set all the ground atoms with predicates connected to the query predicates, where the constants in the query need to be appropriately propagated. This implies that the programs can be hierarchically evaluated in the following way: the models of program Π are $SM(\Pi) = \bigcup_M SM(M \cup \Pi_t)$, for each stable model M in $SM(\Pi_b)$.

The results follows by the fact that for each predicate q in $\mathcal{Q}$, $SM(\Pi)[q] = (SM(\Pi)$ $[rel(\mathcal{Q}, \Pi)])$ $[q]$. In fact, it can be shown that $SM(\Pi)[rel(\mathcal{Q}, \Pi)] = SM(rel(\mathcal{Q}, \Pi) \cup PD_{\mathcal{Q}}))$. Then, for each predicate q in $\mathcal{Q}$, the set of grounds rules having q in the head is in $rel_0(\mathcal{Q}, \Pi) \subseteq rel(\mathcal{Q}, \Pi)$. □

Proof of Proposition 3: We call “coherent” models the models that satisfy the program denials, and “incoherent” the ones that do not. By construction we know that all the models of $\Pi(DB, IC)$ will be coherent and that an incoherent model of $\Pi''(DB, IC)$ will have the atom inc. It is easy to see that $SM(\Pi(DB, IC)) \subseteq SM(\Pi''(DB, IC))$ and that $SM(\Pi''(DB, IC)) \smallsetminus SM(\Pi(DB, IC))$ has only incoherent models. That is because, program $\Pi''(DB, IC)$ has extra models that are incoherent wrt the the program denials in $\Pi(DB, IC)$. In addition, Ans_{coh} is the set of answers for a given query

$\mathcal{Q}$, obtained from the intersection of coherent models of a program $\Pi''(DB, IC))$[7], and Ans_{inc} is the set of answers obtained from the intersection of incoherent models of Π''. Since all the coherent models of $\Pi''(DB, IC)$ are models of $\Pi(DB, IC)$ and since $SM(\Pi(DB, IC)) \subseteq SM(\Pi''(DB, IC))$ we have that Ans_{coh} corresponds exactly to the cautious answers to query $\mathcal{Q}$ obtained from program $\Pi(DB, IC)$. Proving that the cautious answers obtained from $\Pi''(DB, IC)$ are the same as the ones obtained from $\Pi(DB, IC)$ is equivalent to prove that $Ans_{coh} \cap Ans_{inc} = Ans_{coh}$. This implies that the answers from the incoherent models will not affect the results obtained from the coherent models. To prove this, is the same as proving that $Ans_{coh} \subseteq Ans_{inc}$.

By simplicity let us assume we have a query of the form $\mathcal{Q} : Ans(x) \leftarrow P(x)$. Then, $\Pi(\mathcal{Q}) : Ans(x) \leftarrow P\_(x, \mathbf{t}^{\star\star})$, and $\Pi''(\mathcal{Q})$ contains rules: $Ans(x) \leftarrow P\_(x, \mathbf{t}^{\star\star})$; $Ans(x) \leftarrow P\_(x, \mathbf{t}^{\star})$.

Let us suppose for the sake of contradiction, that there exists a tuple $Ans(\bar{c})$ that is in Ans_{coh} and $Ans(\bar{c})$ is not in Ans_{inc}. In order for that to happen, coherent models satisfy both atom $P\_(\bar{c}, \mathbf{t}^{\star\star})$, and $P\_(\bar{c}, \mathbf{t}^{\star})$, and they does not satisfy atom $P\_(\bar{c}, \mathbf{f_a})$. In addition, there exists an incoherent model M_{inc} that satisfies $P\_(\bar{c}, \mathbf{t_a})$, and $P\_(\bar{c}, \mathbf{f_a})$ and such that $Ans(\bar{c})$ is not in M_{inc}. Since $P\_(\bar{c}, \mathbf{t_a})$ is in M_{inc}, then $P\_(c, \mathbf{t}^{\star})$ is in M_{inc}. But, M_{inc} also satisfies the rule $Ans(x) \leftarrow P\_(x, \mathbf{t}^{\star})$ from $\Pi''(\mathcal{Q})$ and therefore $Ans(\bar{c})$ is in M_{inc}. We have reached a contradiction. $\square$

Proof of Proposition 4: Let Π' denotes the ground version of program $\Pi(DB, IC, \mathcal{Q}) \downarrow \mathcal{Q}$. It is easy to see that the ground program $\Pi(DB, IC, \mathcal{Q})$ can be split into a bottom program $\Pi_b = \Pi'$ and a top program $\Pi_t = (\Pi \setminus \Pi_b)$ using as a splitting set [26], all the database facts associated to predicates that are connected with the query predicates in the graph $\mathcal{G}(IC)$. This implies that the programs can be hierarchically evaluated in the following way: the models of program Π are $SM(\Pi) = \bigcup_M SM(M \cup \Pi_t)$, for each stable model M in $SM(\Pi_b)$.

Given the fact that program Π_t does not have rules whose predicates are related with the query predicates, then extensions for the Ans predicate (which collect the answers to query $\mathcal{Q}$) can be obtained by using only the rules from the bottom program Π_b. Then, from the computation of the stable models of program Π, all the models contain the extensions of Ans predicate. Then, we conclude that $\Pi(DB, IC, \mathcal{Q}) \downarrow \mathcal{Q}$ and $\Pi(DB, IC, \mathcal{Q})$ retrieve the same cautious/brave answers to query $\mathcal{Q}$. $\square$

[7] Cautious answers are the tuples obtained from the intersection of all stables models of a repair program.