CQA-WF: Consistent Query Answers to Conjunctive Queries using the Well-Founded Semantics

Julia Belmar Jennifer Cuevas Mónica Caniupán
Universidad del Bío-Bío
Concepción, Chile
{jbelmar,jcuevas,mcaniupa}@ubiobio.cl

***Abstract*—A database instance can become inconsistent with respect to its integrity constraints (ICs), for instance, after update operations. When this happens, it is possible to compute the repairs of the database. A *minimal repair* is a new database instance that satisfies the ICs, is obtained by applying update operations over the original instance, and differs minimally from the original instance. We can evaluate queries over the repairs, an answer to a conjunctive query is *consistent* if it is an answer in every repair. The repairs of database instances can be specified by repair programs. Moreover, we can compute consistent answers to queries by evaluating query programs together with the repair programs under the stable model semantics. The use of logic programs does not exceed the intrinsic complexity of consistent query answering. Nevertheless, for a certain class of conjunctive queries and ICs it is possible to use the alternative *well-founded semantics* (WFS) to evaluate queries. We present *CQA-WF*, a system that allows the computation of consistent answers to conjunctive queries over inconsistent databases with respect to functional dependencies (FDs). *CQA-WF* evaluates logic programs under the WFS. The WFS has lower data complexity than the stable models semantics.**

***Keywords*-Well-founded semantics; inconsistencies, functional dependencies, repairs, consistent query answering;**

I. Introduction

The integrity constraints (ICs) play an important role in databases, because, they capture the intended meaning (semantics) of the data in the database. Nevertheless, it has been reported that databases may become inconsistent with respect to their ICs due to several reasons [1]. We can repair inconsistent databases by performing updates operations over the instances. The notion of database repair was first introduced in [2] ([3] is a good survey in relational repairs). Intuitively, a minimal repair of a database instance D with respect to a set of ICs IC, is a new database instance D' over the same schema of D, that is obtained from D by deleting or inserting whole database tuples, satisfies the ICs, and departs minimally from D under set inclusion. The notion of consistent answer to a first order query was also defined in [2]. A ground tuple $\bar{t}$ is a consistent answer to a query $Q(\bar{x})$ in a database instance D if it is an answer to $Q(\bar{x})$ in every minimal, under set inclusion, repair of D.

Example 1. Consider the database schema $P(x, y)$ and the following functional dependency over P $\psi : x \rightarrow y$. The first two tuples of P violate the functional dependency:

P	
x	y
a	b
a	c
b	d

P'	
x	y
a	c
b	d

P''	
x	y
a	b
b	d

Consistency can be minimally restored by deleting either tuple $P(a, b)$ or tuple $P(a, c)$. Therefore, there are two database repairs P' and P''. Tuple $P(b, d)$ persists in the repairs, since it does not violate the FD ψ. The consistent answer to query $P(x, d)$ is $\{b\}$, since it is the answer in every repair. For the boolean disjunctive query $P(a, b) \vee P(a, c)$

the consistent answer is *yes*, since both database repairs satisfy either one of the tuples in the query. Notice that if we had deleted all the inconsistent data, we would have lost this information. □

In the general case, the data complexity of consistent query answering (CQA) is Π_2^P-complete [1], [3], [4]. Consequently, it is necessary to use an expressive language, such as, disjunctive logic programs under stable models semantics, for which query evaluation has the same data complexity [5]. Disjunctive repair programs with stable models semantics [6], [7] were first introduced in [8], [9] to specify database repairs. Later, simpler and more general repair programs were introduced in [10], [11], [12], [13] for CQA. The logic programming approach works for all universal ICs and queries that are first order or even expressed in Datalog with negation. In [11], [12] it was shown that there is a one to one correspondence between the stable models of the repair program and the database repairs with respect to a set of ICs. Nevertheless, it is possible to identify classes of ICs and queries, for which CQA has lower data complexity. Some polynomial time cases of CQA have been identified in [2], [11], [14], [15]

In this paper we explore the use of the *well-founded semantics* (WFS) of programs [16], [17] to CQA. This semantics has lower data complexity than the stable models semantics, and therefore, it becomes a good alternative for cases in which CQA has lower data complexity. In particular, we present *CQA-WF*, a system that computes consistent query answers to a class of conjunctive queries, with respect to functional dependencies, and that is based in the WFS.

The paper is organized as follows. Section II presents the notion of database repairs, consistent answers to queries and repair programs. Section III presents the WFS and describes the use of the WFS in CQA. Section IV describes the architecture of the system. Finally Section V presents finals remarks and future work.

II. Preliminaries

We consider a relational database schema $\Sigma = (\mathcal{U}, \mathcal{R}, \mathcal{B})$, where $\mathcal{U}$ is the possibly infinite database domain with $null \in \mathcal{U}$, $\mathcal{R}$ is a fixed set of database predicates, each of them with a finite, and ordered set of attributes, and $\mathcal{B}$ is a fixed set of built-in predicates, like comparison predicates, e.g. $\{<, >, =, \neq\}$. $R[i]$ denotes the attribute in position i of predicate $R \in \mathcal{R}$. Database instances of a relational schema Σ are finite collections D of ground atoms of the form $R(c_1, ..., c_n)$, which are called *database tuples*, where $R \in \mathcal{R}$, and $(c_1, ..., c_n)$ is a *tuple* of constants, i.e., elements of $\mathcal{U}$. The extensions for built-in predicates are fixed, and possibly infinite in every database instance.

A functional dependency DF over a relation P is denoted by [14]:

$$\forall \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \bar{x}_5. \neg [P(\bar{x}_1, \bar{x}_2, \bar{x}_4) \wedge P(\bar{x}_1, \bar{x}_3, \bar{x}_5) \wedge \bar{x}_2 \neq \bar{x}_3]. \quad (1)$$

We can also write a functional dependencies as $X \rightarrow Y$, where X and Y are set of attributes of P and $X \cap Y = \emptyset$. The functional dependency on P is satisfied if tuples in P that have the same values in attributes in X, also have the same values for the attributes in Y. If a database fails to satisfy its functional dependencies then, it is *inconsistent* with respect to them.

The semantics of constraint satisfaction in presence of null values we consider is the one defined in [12]. In order to present it, we need to introduce the concept of *relevant* attribute. In presence of functional dependencies only, an attribute is relevant if it appears at least two times in a FD ψ. The set of relevant attributes with respect to a FD ψ is denoted by $\mathcal{A}(\psi)$. For a set of attributes $\mathcal{A}(\psi)$ and a predicate $P \in \mathcal{R}$, $P^{\mathcal{A}}$ denotes the predicate P restricted to the attributes in $\mathcal{A}(\psi)$. $D^{\mathcal{A}}$ denotes the database D with all its database atoms projected onto the attributes in $\mathcal{A}$, i.e., $D^{\mathcal{A}} = \{P^{\mathcal{A}}(\Pi_{\mathcal{A}}(\bar{t})) \mid P(\bar{t}) \in D\}$, where $\Pi_{\mathcal{A}}(\bar{t})$ is the projection on $\mathcal{A}$ of tuple $\bar{t}$. $D^{\mathcal{A}}$ has the same underlying domain $\mathcal{U}$ as D.

A database instance D satisfies a functional dependency $\psi : X \rightarrow Y$ over relation P, denoted by $D \models_N \psi$, if and only if $D^{\mathcal{A}(\psi)} \models \psi^N$, where $\models_N$ denotes satisfaction in presence of null values. This is, D satisfies ψ if any of the relevant attributes

has a $null$ value or the functional dependency is satisfied in the traditional way.

Example 2. Consider the functional dependency ψ: $\forall xyzmn(P(x,y,m) \wedge P(x,z,n) \rightarrow y = z)$ over relation P and the database instance $D = \{P(a,b,a), P(b,c,a),\ P(a,null,d)\}$. The relevant attributes to ψ are x, y, z, since they appear twice in ψ. Then, to verify consistency it is needed to check if $\forall xyz(P^{\mathcal{A}(\psi)}(x,y) \wedge P^{\mathcal{A}(\psi)}(x,z) \rightarrow y = z)$. D is consistent with respect to ψ since, even though there are two tuples that share the value for the first attribute, $P(a,b,a)$ and $P(a,null,d)$, the second tuple has a null value in a relevant attribute. The instance $D' = \{P(a,b,c), P(a,c,d), P(b,c,a)\}$, is inconsistent with respect to ψ through the first two tuples. □

A. Database Repairs and Consistent Query Answers

Inconsistencies with respect to functional dependencies can be restored by deleting tuples. In order to formally define database repairs, we need to introduce the following concept: Let D, D' be database instances over the same schema and domain. The *distance*, $\Delta(D, D')$, between D and D' is the symmetric difference $\Delta(D, D') = (D \setminus D') \cup (D' \setminus D)$ [2].

It is possible to define a partial order between database instances.

Definition 1. [12] Let D, D', D'' be database instances over the same schema and domain $\mathcal{U}$. It holds $D' \leq_D D''$ if and only if:

- For every atom $P(\bar{a}) \in \Delta(D, D')$, with $\bar{a} \in (\mathcal{U} \setminus \{null\})$, it holds that $P(\bar{a}) \in \Delta(D, D'')$.
- For every atom $Q(\bar{a}, \overline{null}) \in \Delta(D, D')$, with $\bar{a} \in (\mathcal{U} \setminus \{null\})$, there exists a $\bar{b} \in \mathcal{U}$ such that $Q(\bar{a}, \bar{b}) \in \Delta(D, D'')$ and $Q(\bar{a}, \bar{b}) \notin \Delta(D, D')$. □

This partial order is used to define the *repairs* of an inconsistent database.

Definition 2. [12] Given a database instance D and a set FD of FDs, a repair of D with respect to FD is a database instance D' over the same schema of D, such that:

(a) $D' \models_N FD$.

(b) D' is $\leq_D$-*minimal* in the class of database instances that satisfy FD with respect to $\models_N$, i.e., there is no database D'' in this class with $D'' <_D D'$, where $D'' <_D D'$ means $D'' \leq_D D'$ but not $D' \leq_D D''$.

The set of repairs of D with respect to FD is denoted by $Rep(D, FD)$. □

A consistent answer to a first order query posed over a possibly inconsistent database D with respect to a set FD of FDs is defined as follows:

Definition 3. [2] Given a database instance D, a tuple $\bar{t}$ is a consistent answer to a query $\mathcal{Q}(\bar{x})$ in D if and only if $\bar{t}$ is an answer to query $\mathcal{Q}(\bar{x})$ in every repair D' of D. Moreover, if a query $\mathcal{Q}$ is an $\mathcal{L}$-sentence, i.e., a boolean query, the consistent answer is *yes* if $\mathcal{Q}$ is true in every repair D' of D; and *no*, otherwise. □

Example 3. Consider the database instance $D = \{P(b,c), P(c,a),\ P(b,e)\}$ and the FD $\psi : x \rightarrow y$ over relation P. D is inconsistent with respect to ψ through tuples $P(b,c)$ and $P(b,e)$. There are two minimal database repairs: $D_1 = \{P(b,c), P(c,a)\}$ and $D_2 = \{P(b,e), P(c,a)\}$. D_1 is obtained by eliminating tuple $P(b,e)$, and D_2 is obtained by eliminating tuple $P(b,c)$. The distances between D and D_1 and D_2 are: $\Delta(D, D_1) = \{P(b,e)\}$, $\Delta(D, D_2) = \{P(b,c)\}$. Note that $D_3 = \{P(c,a)\}$ is also a repair of D, but it is not minimal since $\Delta(D, D_3) = \{P(b,c), P(b,e)\}$ and it holds that $\Delta(D, D_1) \subseteq \Delta(D, D_3)$ and $\Delta(D, D_2) \subseteq \Delta(D, D_3)$. The consistent answer to query $\mathcal{Q}(x) : P(x,y)$ is $\{b, c\}$ □

B. Repair Programs

Database repairs can be specified as stable models (*SM*) of *disjunctive logic programs* [6], [7]. Given an inconsistent database instance D and a set FD of functional dependencies, a disjunctive repair program $\Pi(D, FD)$ is constructed, such that there is a one to one correspondence between the stable models of $\Pi(D, FD)$ and the repairs of D [12].

The repair programs use annotations constants, whose role is to enable the definition of atoms that

can become true in the repairs or false in order to satisfy the FDs. Actually, each atom of the form $P(\bar{a})$ can receive one of the constants in Table I.

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

TABLE I
ANNOTATION CONSTANTS

Definition 4 can be derived from the definition of repair programs given in [12] for general integrity constraints.

Definition 4. The repair program $\Pi(D, FD)$ for a database instance D and set FD of functional dependencies is composed by the following rules:

- For each atom $P(\bar{a}) \in D$ the fact $P(\bar{a})$.
- For every FD ψ of the form (1) the rule: $P\_(\bar{x}_1, \bar{x}_2, \bar{x}_4, \mathbf{f_a}) \vee P\_(\bar{x}_1, \bar{x}_3, \bar{x}_5, \mathbf{f_a}) \leftarrow P\_(\bar{x}_1, \bar{x}_2, \bar{x}_4, \mathbf{t}^\star), P\_(\bar{x}_1, \bar{x}_3, \bar{x}_5, \mathbf{t}^\star), \bar{x}_2 \neq \bar{x}_3, \bar{x}_1 \neq null, \bar{x}_2 \neq null, \bar{x}_3 \neq null$.
- For each predicate $P \in \mathcal{R}$, the annotation rule: $P\_(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x})$.
- For every predicate $P \in \mathcal{R}$, the interpretation rule: $P(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{x}, \mathbf{t}^\star), \ not\ P(\bar{x}, \mathbf{f_a})$. □

Example 4. The repair program $\Pi(D, DF)$ for the functional dependency $\psi : x \to y$ over P, and the database instance $D = \{P(b, c), P(c, a), P(b, e)\}$ in Example 3 has the following rules:

1) $P(b, c).\ P(c, a).\ P(b, e).$
2) $P\_(x, y, \mathbf{f_a}) \vee P\_(x, z, \mathbf{f_a}) \leftarrow P\_(x, y, \mathbf{t}^\star), P\_(x, z, \mathbf{t}^\star), y \neq z, x \neq null,\ y \neq null, z \neq null.$
3) $P\_(x, y, \mathbf{t}^\star) \leftarrow P(x, y).$
4) $P\_(x, y, \mathbf{t}^{\star\star}) \leftarrow P\_(x, y, \mathbf{t}^\star), \ not\ P\_(x, y, \mathbf{f_a}).$

Rules in (1) are the facts. Rule (2) says that either $P(x, y)$ or $P(x, z)$ needs to be eliminated in order to restore consistency of ψ. Rule (3) defines the tuples that are true in the program. Rule (4) defines the tuples that remain true in the repairs.

Program $\Pi(D, DF)$ has two stables models[1]: $\mathcal{M}_1 = \{P\_(b, e, \mathbf{f_a}), P\_(b, c, \mathbf{t}^{\star\star}), P\_(c, a, \mathbf{t}^{\star\star})\}$, $\mathcal{M}_2 = \{P\_(b, c, \mathbf{f_a}), P\_(b, e, \mathbf{t}^{\star\star}), P\_(c, a, \mathbf{t}^{\star\star})\}$. The stable model $\mathcal{M}_1$ says that consistency can be restored by deleting the tuple $P(b, e)$, and then the repair is: $D_1 = \{P(b, c), P(c, a)\}$. The second model says that consistency can be restored by deleting the tuple $P(b, c)$, and the constructed minimal repair is $D_2 = \{P(b, e), P(c, a)\}$. □

In order to use repair programs to compute consistent answers, queries have to be 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 [18], and next replacing every positive literal $P(\bar{s})$ by $P(\bar{s}, \mathbf{t}^{\star\star})$, and every negative literal $not\ P(\bar{s})$ by $not\ P(\bar{s}, \mathbf{t}^{\star\star})$. Thus, in order to get consistent answers, we need to create the program $\Pi(\mathcal{Q}) \cup \Pi(D, FD)$, and evaluate it under the stable models semantics with cautious reasoning [6], [7].

Example 5. Query $\mathcal{Q}(x) : P(x, y)$ in Example 3 is transformed into the Datalog program $\Pi(\mathcal{Q}) : Ans(x) \leftarrow P\_(x, y, \mathbf{t}^{\star\star})$. There are two stable models of $\Pi(D, FD) \cup \Pi(\mathcal{Q})$: $\mathcal{M}_1 = \{P\_(b, e, \mathbf{f_a}), P\_(b, c, \mathbf{t}^{\star\star}), P\_(c, a, \mathbf{t}^{\star\star}), Ans(b), Ans(c)\}$, $\mathcal{M}_2 = \{P\_(b, c, \mathbf{f_a}), P\_(b, e, \mathbf{t}^{\star\star}), P\_(c, a, \mathbf{t}^{\star\star}), Ans(b), Ans(c)\}$. Therefore, the consistent answer to $\mathcal{Q}$ is $\{b, c\}$ as expected. □

III. WELL-FOUNDED SEMANTICS AND CQA

A. Well-founded Semantics

The *well-founded* semantics for normal logic programs was introduced in [16], and later extended to disjunctive logic programs [19], [20]. It has been used as an alternative to the stable models semantics [21], [6], [7]. In fact, if a general logic program has a total well-founded model, that model is the unique stable model [16]. We adopt the framework presented in [19], that defines the WFS in terms of an operator that maps interpretations to interpretations, obtaining a *well-founded interpretation* (WFI) as a least fixpoint.

The WFI for a ground disjunctive program Π consists of three disjoint and complementary sets of

[1]For simplification purposes, the stable models are restricted to the atoms annotated with $\mathbf{f_a}$ and $\mathbf{t}^{\star\star}$.

ground atoms: $W_\Pi = \langle W^+, W^-, W^u \rangle$, where W^+ is the set of true atoms, W^- is the set of false atoms, and W^u is the set of undetermined atoms [19]. The WFI is defined as the least fixpoint of an operator $\mathcal{W}_\Pi$, that is a mapping between interpretations of the form $I = \langle I^+, I^-, I^u \rangle$, with I^+, I^-, I^u disjoint sets of ground atoms that cover the whole Herbrand base of the program [21].

The stable models semantics tries to find alternative models for the program (possibly more than one), giving to all atoms a true or false value. Therefore, a program can have several alternative stable models, but only one WFI. The stable models can be computed from the WFI by, starting from the atoms which are true or false, trying to give different values to the unknown atoms. Therefore, the set W^+ of true atoms of the WFI is contained in every stable model of the program, and the set of false atoms of the WFI is a subset of the set of atoms that are false in every stable model [19], [16].

If we define a *literal* as a formula of the form A or $not\ A$, with A atomic, then interpretations I can be represented as sets of ground literals. In this case, I^+ is the set of atoms (i.e., positive literals) in I, and I^- is $not.\{\ not\ A \mid \ \ not\ A \in I\}$,[2] and $I^u = \{A \mid A$ is ground atom and both $A,\ not\ A \notin I\}$. Then, I^u becomes implicitly the set of atoms A such that neither A nor $not\ A$ can be found in $I^+ \cup\ not.I^-$.

The operator $\mathcal{W}_\Pi$ is based in an extension of the notion of *unfounded set* to disjunctive programs. Unfounded sets contain atoms that are definitely not derivable from a given program with respect to a given interpretation, and as a consequence, they are declared false.

Definition 5. [19] Let I be an interpretation for (the ground version of a) program Π. A set $X \subseteq B_\Pi$ of ground atoms is an *unfounded set* for Π with respect to I if for each $a \in X$ (an unfounded atom in X), for each rule $r \in \Pi$ (the instantiated i.e., ground version of Π), such that $a \in H(r)$, the head of rule r, at least one of the following conditions holds:

(a) $B(r) \cap\ not.I \neq \emptyset$, i.e., the body of r is false regarding I.
(b) $B^+(r) \cap X \neq \emptyset$, i.e., some positive body literal belongs to X.
(c) $(H(r) \setminus X) \cap I \neq \emptyset$, i.e., an atom in the head of r, distinct from a and other elements in X, is true with respect to I. □

The union of all the unfounded sets for a program Π regarding an interpretation I, $GUS_\Pi(I)$, is called the *greatest unfounded set with respect to* I. For normal programs, the GUS is also an unfounded set, but for disjunctive programs this might not be the case [19]. However, it has been shown that for our repair programs GUS is an unfounded set [22].

Definition 6. Given a ground disjunctive program Π, the *well-founded operator* (WFO), denoted by $\mathcal{W}_\Pi$, is defined on interpretations I for which $GUS_\Pi(I)$ is unfounded, by:

$$\mathcal{W}_\Pi(I) := \Gamma_\Pi(I)\ \cup\ not.GUS_\Pi(I),$$

where $\Gamma_\Pi(I)$ is the immediate consequence operator that declares an atom A true with respect to I if there exists a rule in Π, such that A is in the head of the rule, the body of the rule is true with respect to I, and the other atoms in the head of the rule (if any) are false with respect to I. □

The *well founded interpretation* of a ground program Π is defined as the fixpoint W_Π of the interpretations defined by: $W_0 := \emptyset$, $W_{k+1} := \mathcal{W}_\Pi(W_k)$. W_Π can be computed in polynomial time [19].

Example 6. The WFI for the repair program in Example 4 is[3]: (i) $W^+ = \{P\_(c, a, \mathbf{t}^\star)\}$. (ii) $W^u = \{P\_(b, e, f_a), P\_(b, e, \mathbf{t}^\star), P\_(b, c, \mathbf{f_a}), P\_(b, c, \mathbf{t}^\star)\}$. (iii) W^- is composed for combinations of atoms of the form $P\_(c_1, c_2, \mathbf{f_a})$ and $P\_(c_1, c_2, \mathbf{t}^\star)$. □

[2]Remark $not.\mathcal{L}$, with $\mathcal{L}$ a set of literals, is the set of literals that are complementary to those in $\mathcal{L}$. For a literal L, $not.L$ denotes the literal that is complementary to L.

[3]For simplification purposes, the WFS is restricted to the atoms annotated with $\mathbf{t}^\star$, $\mathbf{f_a}$ or *Ans* atoms.

B. Using the WFI of Programs in CQA

In this section we use the concepts of safe database and conflict closure of a database, which were presented in [13]. The *safe database* is the portion of the database that does not participate in any violation of FDs, and that will never be touched by the repair process. The *conflict closure of a database* is the set of tuples that violate the FDs or are going to be changed to avoid new violations of FDs. According to [22] we are able to capture the safe and affected database by using W^+ and W^u of the WFI of program $\Pi(D, FD)$, when considering only one functional dependency or key dependency per relation. Therefore, as in [13] we could compute database repairs by computing the repairs for the affected portion of data, and then combining them with the safe portion of data. However, we are not interested in computing repairs, but in retrieving consistent answers to queries. Thus, what we want is to use as much as possible sets W^+ and W^u of the WFI of program $\Pi(D, FD)$, to compute consistent answers to conjunctive queries with respect to FDs.

In order to use the WFI of programs for CQA we have to restrict ourselves to the restricted classes of conjunctive queries of the form (2) and (3) [14].

$$Ans(w_1 \dots w_m) \leftarrow \exists z_1 \dots z_n (P_1(\bar{x}_1), \dots P_n(\bar{x}_n)),^4 \quad (2)$$

where $w_1 \dots w_m, z_1 \dots z_n$ are all the variables that appear in the atoms of the body of the query, each $\bar{x}_i$ matches the arity of P_i, variables $w_1 \dots w_m$ are the *free* variables of the query. The query is called *simple* if there are no constants and no repeated symbols in the query (no joins between relations are allowed).

$$Ans \leftarrow \exists z_1 \dots z_n (P_1(\bar{x}_1), \dots P_n(\bar{x}_n)), \quad (3)$$

where $z_1 \dots z_n$ are variables that appear in the atoms of the query, and $\bar{x}_1 \dots \bar{x}_n$ are variables and/or constants, and no joins between relations are allowed.

[4]Usually the existential quantifiers are implicit on the query, but in this section, we will write them explicitelly on queries.

Definition 7. [22] For a database instance D, set FD of FDs, and a conjunctive query $\mathcal{Q}$ of the form (2), a tuple $\bar{t}$ is a *well-founded answer* to $\mathcal{Q}$ if $Ans(\bar{t})$ is in $W^+ \cup W^u$ of the WFI of program $\Pi(D, FD, \mathcal{Q})$. The set of *well-founded answers* to $\mathcal{Q}$ are denoted by $WFA^{+u}(\mathcal{Q})$. □

The set W^u of the WFI of programs can only be used directly for CQA when the attributes projected in the query are attributes in the antecedent of some FD, because in this case, the inconsistent tuples from W^u will have the same value for these attributes. Therefore, we can use the WFI of programs directly to compute consistent answers to queries of the form (2) when each of the free variables on it refer to attributes in the antecedent of a FD.

Definition 8. [22] Given a relation P with FD $X \rightarrow Y$, we can divide the attributes of P into: (i) *antecedent attributes*, i.e., the attributes in X. (ii) *consequent attributes*, i.e., the attributes in Y. (iii) *simple attributes*, which are attributes neither in X nor Y. Moreover, for a query $\mathcal{Q}$ of the form (2), we say that a free variable w_i with $w_i \in \{w_1 \dots w_m\}$ refers to an antecedent (respectively, consequent, simple) attribute, if w_i matches a variable in the position of an antecedent (respectively, consequent, simple) attribute on a query predicate P. □

It was proven in [22] that for a database instance D, set FD of FDs (at most one functional or key dependency per relation), if the free variables in query $\mathcal{Q}$ of the form (2) refer to antecedent attributes, then the consistent answers to $\mathcal{Q}$ with respect to FD coincide with the well-founded answers to $\mathcal{Q}$.

Example 7. Consider the relation $A(id, name, age)$, $D = \{A(1, john, 11), A(1, mary, 20), A(2, mary, 20)\}$, FD $\psi : id \rightarrow name$, and query $\mathcal{Q} : Ans(x) \leftarrow \exists yz A(x, y, z)$, where x represents an antecedent attribute. The sets W^+ and W^u of the WFI of program $\Pi(D, DF, \mathcal{Q})$ are: $W^+ = \{A\_(2, mary, 20, \mathbf{t}^\star\star), Ans(2)\}$. $W^u = \{A\_(1, john, 11, \mathbf{t}^\star\star), A\_(\overline{1, mary}, 20, \mathbf{t}^\star\star), Ans(1)\}$. The well-founded answer is $\overline{WFA^{+u}}(\mathcal{Q}) = \{1, 2\}$, that coincides with the consistent answer to $\mathcal{Q}$. □

There are cases where we need to rewrite queries in such a way that, when they are evaluated on W^u of $\Pi(D, FD)$, they only retrieve consistent answers. We illustrate this in the example below.

Example 8. For $\Sigma = \{R(x, y, z, w)\}$, FD: $z \rightarrow w$, and $D = \{R(a, b, c, d),\ R(a, b, c, e),\ R(b, c, d, e)\}$. For the repair program $\Pi(D, FD)$, $W^+ = \{R\_(b, c, d, e, \mathbf{t}^{\star\star})\}$, and $W^u = \{R\_(a, b, c, d, \mathbf{t}^{\star\star}), R\_(a, b, c, e, \mathbf{t}^{\star\star})\}$. There are two repairs: $\{R(b, c, d, e),\ R(a, b, c, d)\}$ and $\{R(b, c, d, e), R(a, b, c, e)\}$.

For $\mathcal{Q}$: $Ans(z, w) \leftarrow \exists xy R(x, y, z, w)$, the free variable z refers to the attribute in the antecedent of the FD, and variable w refers to its consequent. For $\Pi(D, FD, \mathcal{Q})$ it holds $W^+ = \{Ans(d, e)\}$, and $W^u = \{Ans(c, d),\ Ans(c, e)\}$. Therefore, $WFA^{+u}(\mathcal{Q})$ are $\{(d, e),\ (c, d),\ (c, e)\}$, but the consistent answer to $\mathcal{Q}$ is $\{(d, e)\}$. Thus, in this case, W^u of $\Pi(D, FD, \mathcal{Q})$ does not provide consistent answer to the query. However, we can filter the inconsistent tuples from W^u of $\Pi(D, FD)$ by evaluating the following query on it: $\mathcal{Q}'$: $Ans(z, w) \leftarrow \exists xy R\_(x, y, z, w, \mathbf{t}^{\star\star}) \wedge \forall x'y'w'(R\_(x', y', z, w', \mathbf{t}^{\star\star}) \rightarrow w' = w)$. When $\mathcal{Q}'$ is evaluated on W^u of $\Pi(D, FD)$, the answer is empty. Therefore, the final answer to $\mathcal{Q}$ is $\{(d, e)\}$ as expected.

For query $\mathcal{Q}$: $Ans(z, x) \leftarrow \exists yw R(x, y, z, w)$, where the free variable z refers to the attribute in the antecedent of the FD, and variable x refers to a simple attribute, W^+ of $\Pi(D, FD, \mathcal{Q})$ is $\{Ans(d, b)\}$. Since z refers to an antecedent attribute, we know that tuples in W^u that are inconsistent with respect to FD will share the value for that attribute. Thus, we just need to ensure that tuples from W^u have the same value for the attribute referenced by variable x. Therefore, $\mathcal{Q}'$ is: $Ans(z, x) \leftarrow \exists yw R\_(x, y, z, w, \mathbf{t}^{\star\star}) \wedge \forall x'y'w'(R\_(x', y', z, w', \mathbf{t}^{\star\star}) \rightarrow x' = x)$. The answer to $\mathcal{Q}'$ evaluated on W^u of $\Pi(D, FD)$ is $\{(c, a)\}$. Therefore, the final well-founded answers to $\mathcal{Q}$ are $\{(d, b), (c, a)\}$, which coincide with the consistent answers to $\mathcal{Q}$.

Moreover, for $\mathcal{Q}$: $Ans(y) \leftarrow \exists xzw\ R\_(x, y, z, w)$, where variable y refers to a simple attribute, W^+ of $\Pi(D, FD, \mathcal{Q})$ is $\{Ans(c)\}$. Here we just need to ensure that the value for the attribute referenced by variable y is the same in every inconsistent tuple in W^u. Hence, $\mathcal{Q}'$ is: $Ans(y) \leftarrow \exists xzw R\_(x, y, z, w, \mathbf{t}^{\star\star}) \wedge \forall x'y'z'w'(R\_(x', y', z', w', \mathbf{t}^{\star\star}) \rightarrow \bar{y}' = y)$. The answer to $\mathcal{Q}'$ evaluated on W^u of $\Pi(D, FD)$ is $\{b\}$. Therefore, the well-founded answers to $\mathcal{Q}$ are $\{c, b\}$, which coincide with the consistent answers to $\mathcal{Q}$. □

The situation is not different for boolean conjunctive queries of the form (3). For this kind of queries we also need a new, rewritten query in order to retrieve consistent answers from the set of undetermined atoms of the WFI of a repair program. However, the new query is only needed when it is not possible to compute an answer from W^+ of $\Pi(D, FD, \mathcal{Q})$. We illustrate this in the example below.

Example 9. (Example 8 cont.) For $\mathcal{Q}$: $Ans \leftarrow \exists xzw\ \ R(x, c, z, w)$, W^+ of $\Pi(D, FD, \mathcal{Q})$ is $\{Ans\}$, therefore the well-founded answer is *yes*, which coincides with the consistent answer. For $\mathcal{Q}$: $Ans \leftarrow \exists yzw\ \ R(a, y, z, w)$, W^+ of $\Pi(D, FD, \mathcal{Q})$ does not have an Ans-atom, then we need to check if we can obtain an answer from W^u of the repair program. The rewritten query is: $\mathcal{Q}'$: $Ans \leftarrow \exists yzw R\_(a, y, z, w, \mathbf{t}^{\star\star}) \wedge \forall x'y'z'w'(R\_(x', y', z', w', \mathbf{t}^{\star\star}) \rightarrow x' = a)$. When query $\mathcal{Q}'$ is evaluated on $W^u = \{R\_(a, b, c, d, \mathbf{t}^{\star\star}), R\_(a, b, c, e, \mathbf{t}^{\star\star})\}$, the answer is *yes*, since every inconsistent tuple with respect to the FD has the same value for the first attribute. Therefore, the well-founded answer is *yes*, and coincides with the consistent answer to $\mathcal{Q}$.

Moreover, for $\mathcal{Q}$: $Ans \leftarrow \exists xyz\ \ R(x, y, z, f)$, there is no Ans-atom in W^+ of $\Pi(D, FD, \mathcal{Q})$, then the following rewritten query is generated: $\mathcal{Q}'$: $Ans \leftarrow \exists xyz R\_(x, y, z, f, \mathbf{t}^{\star\star}) \wedge \forall x'y'z'w'(R\_(x', y', z', w', \mathbf{t}^{\star\star}) \rightarrow w' = f.)$ However, the answer to $\mathcal{Q}'$ is also negative in W^u, hence the well-founded answer to $\mathcal{Q}$ is *no*, which coincides with the consistent answer to $\mathcal{Q}$. □

IV. System

CQA-WF is a web system that computes consis-

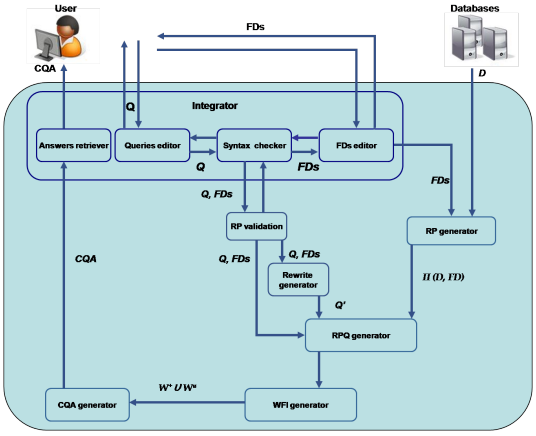

Fig. 1. *CQA-WF*'s architecture.

Rewritten Query Generation$(FD, \mathcal{Q})$

set of variables: *FANS*, *AV*, *CV*, *AVP*, *CVP*;
$\Pi(\mathcal{Q}) := GenerateQueryProgram(\mathcal{Q})$;
$\mathcal{Q}_{rew} := \Pi(\mathcal{Q})$;
$FANS := IdentifyFreeVariables(\mathcal{Q})$;
$AV := IdentifyVariablesInAntecedentsFDs(\mathcal{Q}, FD)$;
$CV := IdentifyVariablesInConsequentsFDs(\mathcal{Q}, FD)$;
For each $P(\bar{x}) \in \mathcal{Q}$ **do**
  $AVP := \bar{x} \cap AV$;
  $CVP := \bar{x} \cap CV$;
  $FVP := \bar{x} \cap FANS$;
  if $(\{AVP \cup CVP \cup FVP\} \neq \emptyset)$ **then**
    $\bar{y} := \bar{x} \setminus AVP$;
    $\bar{y} := GenerateFreshVariables(\bar{y})$;
    $\mathcal{Q}_{rew} := \mathcal{Q}_{rew} \wedge \forall \bar{y} GenerateAtom(P(\bar{x}), \bar{y}, AVP, CVP, FVP)$;
  end if
end for
return $\mathcal{Q}_{rew}$

Fig. 2. Algorithm for queries of the form (2)

tent answers to conjunctive queries of form (2) and (3) based in the well-founded interpretation of logic programs. *CQA-WF* uses an algorithm implemented in PROLOG language[5] to compute sets W^+ and W^u of the WFI of programs. This algorithm works for Datalog programs with negation but without disjunction.[6] Repair programs are disjunctive programs, however, when we consider only functional dependencies we can translated the disjunctive repair programs into non-disjunctive programs (see [22] for details). Therefore, *CQA-WF* generates repair programs free of disjunction, these programs are equivalent to the original programs in terms of their stable models.

Figure 1 shows the architecture of the system. *CQA-WF* allows the edition of functional dependencies and queries. The system checks syntax of FDs and queries, generates rewritten queries, repair programs and query programs, and also retrieves well-founded (consistent) answers to queries. The system is composed by the following modules:

1) *FDs editor:* that allows the edition of FDs.
2) *Queries editor:* that permits to write queries.
3) *Syntax checker:* that verifies the syntax of FDs and queries.
4) *RP validation:* that verifies that the system has all the needed data to construct logic programs.
5) *Rewrite generator:* that generates rewritten queries.
6) *RP generator:* that generates repair programs.
7) *RPQ generator:* that generates query programs.
8) *WFI generator:* that computes the relevant sets W^+ and W^u of the WFI of programs.
9) *CQA generator:* that gets well-founded answers to queries.
10) *Answers retriever:* that retrieves the consistent answers to queries.

Figure 2 presents the algorithm that generates a rewritten query $\mathcal{Q}_{rew}$ for a given query $\mathcal{Q}$ of the form (2). The input to the algorithm consists of the set FD of FDs, and the query $\mathcal{Q}$. The algorithm first identifies the free variables in $\mathcal{Q}$. After that, it determines which variables refer to attributes in FDs. Then, for each atom in the query that shares variables with the Ans predicate, an atom in the rewritten query is generated. A tuple $\bar{t}$ is an answer to the rewritten query if tuples from set W^u of $\Pi(D, FD)$ have the same values for the attributes referenced by the free variables in the query.

Figure 3 shows the algorithm that generates a rewritten query $\mathcal{Q}_{rew}$ for a query $\mathcal{Q}$ of the form (3). The algorithm first identifies the query predicates that have instantiated variables. Then for each of

[5]http://www.swi-prolog.org/

[6]Algorithm reported in *"A Tiny Interpreter for Datalog with Well-Founded"*, Bertram Ludäsher, 1995.

Rewritten Query Generation($\mathcal{Q}$)

set of constants: *CONS*;
$\Pi(\mathcal{Q}) := GenerateQueryProgram(\mathcal{Q})$;
$\mathcal{Q}_{rew} := \Pi(\mathcal{Q})$;
$CONS := IdentifyConstants(\mathcal{Q})$;
For each $P(\bar{x}) \in \mathcal{Q}$ **do**
  if $(\{CONS \cap \bar{x}\} \neq \emptyset)$ **then**
    $\bar{y} := \bar{x} \setminus \{CONS \cap \bar{x}\}$;
    $\bar{y} := GenerateFreshVariables(\bar{y})$;
    $\mathcal{Q}_{rew} := \mathcal{Q}_{rew} \wedge \forall \bar{y} GenerateAtom(P(\bar{x}), \bar{y}, CONS)$;
  end if
end for
return $\mathcal{Q}_{rew}$

Fig. 3. Algorithm for queries of the form (3)

those query atoms, the algorithm generates a corresponding atom in the rewritten query. The answer to $\mathcal{Q}_{rew}$ will be *yes* if tuples from W^u of $\Pi(D, FD)$ have the same values for the instantiated variables in the query.

V. Conclusiones

We presented *CQA-WF*, a system to compute consistent answers to queries based in the well-founded semantics of logic programs. The WFS has been used as an alternative to the stable model semantics for Datalog programs with negation. The well-founded interpretation of logic programs can be computed in polinomial time, then it is relevant to use it in CQA since the high cost of CQA for general queries and integrity constraints.

We have considered an important class of conjunctive queries with and without projection. Our results generalize some preliminary results obtained in [8] for CQA (for a different kind of repair programs).

CQA-WF is the first system that implements repair programs under the well-founded semantics. In [23] a more general system is presented, it works for general integrity constraints and computes consistent answers to general conjunctive queries under the stable model semantics. Other polinomial time systems to compute consistent answers are: (i) *Queca* system [24] that implements query rewritten for conjunctive queries without projection and works for universal integrity constraints. (ii) *Hippo* system [14] that also implements query rewritten for first conjunctive queries without projections, it is based in graph methods, and works for FDs (at most one per relation). (iii) *ConQuer* system [25] that allows to obtain consistent answers to conjunctive queries with projection and joins between different database relations with respect to primary keys. We left as a future work the comparison of *CQA-WF* with the polinomial time reported systems.

The rewriting of queries we introduced corresponds to the rewriting method presented in [15] and implemented in [25]. This method works for a more general case of conjunctive queries, the *C-Tree* queries, which allow joins between different database relations. In [15] the rewritten query is evaluated directly on the inconsistent database instance. In this manner, its answers correspond to the consistent answers to the original query. Here, the rewritten query filters inconsistent answers from set W^u of the WFI of a repair program. Thus, the rewritten query is evaluated on a subset of the database, the portion of data that falls in W^u. Therefore, we compute rewritten queries on small portions of the database, instead of processing them on the original database.

As reported in [22], the WFI of programs can also be used to compute approximate answers to positive Datalog queries and general ICs.

Acknowledgments

Mónica Caniupán is funded by University of Bío-Bío (Grant DIUBB 110115 2/R).

References

[1] L. Bertossi and J. Chomicki, "Query Answering in Inconsistent Databases," in *Logics for Emerging Applications of Databases*, ser. Springer LNCS 1973, 2003, pp. 43–83.

[2] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent Query Answers in Inconsistent Databases," in *Proceedings of the ACM Symposium on Principles of Database Systems(PODS'99)*, 1999, pp. 68–79.

[3] L. Bertossi, "Consistent Query Answering in Databases," *ACM Sigmod Record*, vol. 35, no. 2, pp. 68–76, 2006.

[4] C. H. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1994.

[5] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, "Complexity and expressive power of logic programming," *ACM Comput. Surv.*, vol. 33, no. 3, pp. 374–425, 2001.

[6] M. Gelfond and V. Lifschitz, "Classical Negation in Logic Programs and Disjunctive Databases," *New Generation Computing*, vol. 9, no. 3/4, pp. 365–386, 1991.

[7] T. C. Przymusinski, "Stable Semantics for Disjunctive Programs," *New Generation Computing*, vol. 9, pp. 401–424, 1991.

[8] M. Arenas, L. Bertossi, and J. Chomicki, "Answer Sets for Consistent Query Answering in Inconsistent Databases," *Theory and Practice of Logic Programming*, vol. 3, no. 4-5, pp. 393–424, 2003.

[9] G. Greco, S. Greco, and E. Zumpano, "A Logical Framework for Querying and Repairing Inconsistent Databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1389–1408, 2003.

[10] P. Barceló and L. Bertossi, "Logic Programs for Querying Inconsistent Databases," in *Proceedings of the International Symposium on Practical Aspects of Declarative Languages (PADL'03)*, ser. Springer LNCS 2562, 2003, pp. 208–222.

[11] P. Barceló, L. Bertossi, and L. Bravo, "Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets," in *Semantics in Databases*, ser. Springer LNCS 2582, 2003, pp. 1–27.

[12] L. Bravo and L. Bertossi, "Semantically Correct Query Answers in the Presence of Null Values," in *Proceedings of the EDBT WS on Inconsistency and Incompleteness in Databases (IIDB'06)*, ser. Springer LNCS 4254, 2006, pp. 336–357.

[13] T. Eiter, M. Fink, G. Greco, and D. Lembo, "Efficient Evaluation of Logic Programs for Querying Data Integration Systems," in *Proceedings of the International Conference on Logic Programming (ICLP'03)*, ser. Springer LNCS 2916, 2003, pp. 163–177.

[14] J. Chomicki and J. Marcinkowski, "Minimal-Change Integrity Maintenance using Tuple Deletions," *Information and Computation*, vol. 197, no. 1-2, pp. 90–121, 2005.

[15] A. Fuxman and R. J. Miller, "First-order query rewriting for inconsistent databases." in *ICDT*, 2005, pp. 337–351.

[16] A. Van Gelder, K. Ross, and J. S. Schlipf, "Unfounded Sets and Well-Founded Semantics for General Logic Programs," in *Proceedings of the ACM Symposium on Principles of Database Systems(PODS'88)*, 1988, pp. 221–230.

[17] P. R. Nicola Leone and F. Scarcello, "Disjunctive stable models: unfounded sets, fixpoint semantics, and computation," *Inf. Comput.*, vol. 135, pp. 69–112, 1997.

[18] J. W. Lloyd, *Foundations of logic programming; (2nd extended ed.)*. Springer-Verlag New York, Inc., 1987.

[19] N. Leone, P. Rullo, and F. Scarcello, "Disjunctive stable models: unfounded sets, fixpoint semantics, and computation," *Inf. Comput.*, vol. 135, no. 2, pp. 69–112, 1997.

[20] T. Przymusinski, "Well-founded semantics coincides with three-valued stable semantics," *Fundam. Inf.*, vol. 13, no. 4, pp. 445–463, 1990.

[21] M. Gelfond and V. Lifschitz, "The Stable Model Semantics for Logic Programming," in *Proceedings of the International Conference on Logic Programming (ICLP'88)*, 1988, pp. 1070–1080.

[22] M. Caniupán, "Optimizing And Implementing Repair Programs For Consistent Query Answering In Databases," Ph.D. dissertation, School of Computer Science, Carleton University, 2007. [Online]. Available: http://www.face.ubiobio.cl/~mcaniupa/

[23] M. Caniupán and L. E. Bertossi, "The consistency extractor system: Answer set programs for consistent query answering in databases," *Data Knowl. Eng.*, vol. 69, no. 6, pp. 545–572, 2010.

[24] A. Celle and L. Bertossi, "Querying Inconsistent Databases: Algorithms and Implementation," in *Proceedings of the First International Conference on Computational Logic*. Springer-Verlag, 2000, pp. 942–956.

[25] A. Fuxman, E. Fazli, and R. J. Miller, "ConQuer: Efficient Management of Inconsistent Databases," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 155–166.