ECAI 2010
H. Coelho et al. (Eds.)
IOS Press, 2010

doi:10.3233/978-1-60750-606-5-125

An Argumentation-based Approach to Database Repair

Emanuel Santos[1 2] and **João Pavão Martins** [2] and **Helena Galhardas** [2]

Abstract. The access to *high-quality* data is essential to make accurate decisions. Consequently, when a database becomes inconsistent is crucial to restore its consistency. The main approach for database consistency restoration is based on the notion of *repair*. In this work, we use *Argumentation* techniques to provide a better understand of the reasoning process behind database reparation. We introduced an extended argumentation framework that provides a comprehensive and alternative way to identify, represent and resolve the conflicts between tuples in an inconsistent database. We studied the complexity of this framework and show that it can be used to check the optimality of a repair based on the extended notions of locally, semi-globally and globally optimal repair with respect to Denial Constraints and Tuple-Generating Dependencies classes.

1 Introduction

In order to make accurate decisions, information systems users need to access high-quality data, i.e., without errors, duplicates or inconsistencies. One of the aims of Data Cleaning [5] is to restore the consistency of an inconsistent database by maximizing its *quality*. In this context, the notion of *repair* [3] was introduced: a consistent database that results from applying repair operations (typically, deletion, insertion or modification of tuples) in the original (inconsistent) database. Depending on the *repair model* [9] used, different notions of *repair* can be implemented. In addition, different notions of optimal repair have been introduced [13].

Argumentation [6] is a comprehensive process that allows us to explain and justify a conclusion derived in a reasoning process, to identify conflicts and find the pros and cons for a given conclusion through a alternating use of arguments and counter-arguments. It involves one or more entities (e.g., users, DBMS, agents, etc.) that independently interact and exchange arguments with one another. Many frameworks for modelling argumentation in logic have been presented (e.g., [2, 7, 6, 11]). Each one introduces a formal representation of arguments and evaluation techniques to settle possible *conflicts* between them.

We introduce an argumentation framework that provides a comprehensive and alternative way to identify, represent and resolve the conflicts between tuples in an inconsistent database. We extend the monological argument framework introduced in [12]. We studied the complexity of this framework and show that it can be used to check the optimality of a repair with respect to Denial Constraints and Tuple-Generating Dependencies classes. For this purpose we extend the notions of locally, semi-globally and globally optimal repair introduced in [13]. Plan of the paper: Section 2 introduces notation, integrity constraints classes and conflicting set, repair and optimal repair definitions; Section 3 introduces the argumentation framework, definition of argument (Section 3.1), counter-argument (Section 3.2) and argumentation tree (Section 3.3); Section 4 presents results regarding the optimal repair checking; Section 5 shows conclusions and future work. Due to space limitations, all the proofs are omitted. A extended version of this paper can be found in [1].

[1] Phd Student supported by Fundação para a Ciência e Tecnologia under PhD grant SFRH/BD/27253/2006.
[2] Instituto Superior Técnico, Technical University of Lisbon, Portugal, email: {esantos, joao.pavao.martins, helena.galhardas}@ist.utl.pt.

2 Preliminaries

We assume the base definitions of the relational model found in [4]. We use $X, Y, ...$ to denote sets of attributes, $R, S, ...$ to denote relation schemas, $\mathbf{R}, \mathbf{S}, ...$ to denote database schemas, $\mathbf{r}, \mathbf{s}, ...$ to denote database instances, $r, s, ...$ to denote relations and $t, x, y, ...$ to denote tuples. An apostrophe or an index can also be added to each of the previous representation letters. We denote by $t[X]$ the attribute value of t wrt the attribute X. We represent a database instance $\mathbf{r} = \{r_1, ..., r_n\}$, with a relation schema $\mathbf{R} = \{R_1(X_1), ..., R_n(X_n)\}$, as a set of tuples of the form $R_i(t_j)$, where $R_i(t_j)$ denotes a tuple t_j with scheme R_i. Thus, the set operations over database instances are well defined.

2.1 Inconsistency

We consider the most broad integrity constraint classes that are commonly associated to databases:

1. **Denial Integrity Constraints** (DC), denoted as $[(R_1, ..., R_n), \phi]$:

$$\forall_{t_1,..,t_n}.(R_1(t_1) \wedge ... \wedge R_n(t_n)) \Rightarrow \phi(t_1, ..., t_n),$$

where ϕ is a propositional formula of comparison atoms $t_i[X] = t_j[Y]$, $t_i[X] \neq t_j[Y]$, $t_i[X] < t_j[Y]$ or $t_i[X] \leq t_j[Y]$, where $1 \leq i, j \leq n$.

2. **Tuple-Generating Dependencies** (TGD), denoted as $[(R_1, ..., R_m), \phi_1 \sqsubseteq (R_{m+1}, ..., R_n), \phi_2]$:

$$\forall_{t_1,..,t_m}. \bigwedge_{i=1}^{i=m} R_i(t_i) \wedge \phi_1 \Rightarrow \exists_{t_{m+1},...,t_n}. \bigwedge_{i=m+1}^{i=n} R_i(t_i) \wedge \phi_2$$

where ϕ_1 and ϕ_2 are conjunctions of equality atoms $t_i[X] = t_j[Y]$, where $1 \leq i, j \leq m$ and $1 \leq i, j \leq n$, respectively.

Notice that Functional, Inclusion, Key and Foreign Key Dependencies are special cases of the previous classes [4]. We say that a set Σ of integrity constraints is *acyclic* if its dependency graph has no cycles. The dependency graph of Σ is the graph whose nodes are the relation symbols occurring in Σ and whose edges are pairs of the

form (R, S) such that R occurs in the left-hand side of some TGD in Σ and S occurs in the right-hand side of that TGD.

For a given integrity constraint γ, we denote by $[\gamma]_l$ and $[\gamma]_r$ the list of all relation symbols that occur in the left-hand side and in the right-hand side of γ, respectively. Moreover, we extend these denotations for a given set of integrity constraints.

Given a database instance $\mathbf{r}$ of $\mathbf{R}$ and a set of integrity constraints Σ, we say that $\mathbf{r}$ is *consistent* wrt Σ (or that $\mathbf{r}$ *satisfies* Σ) if $\mathbf{r} \models \gamma$ for every $\gamma \in \Sigma$, i.e. $\mathbf{r} \models \Sigma$, in the standard model-theoretic sense; otherwise, we say that $\mathbf{r}$ is *inconsistent* wrt Σ (or that $\mathbf{r}$ violates Σ).

In addition to the original database, we have a possibly infinite set of tuples, called *extra tuples*, which can be added to the database.

Figure 1. The database $\mathbf{r}_{ex}$ and the respective set of extra tuples $\mathbf{r}_{ex}^{ext}$.

$workers$

	Name	Depart	SLevel
t_1	Adrian	d_A	3
t_2	Peter	d_A	3
t_3	Peter	d_A	5
t_4	Adrian	d_B	4
t_5	Peter	d_B	5
t_6	Peter	d_B	6
t_7	Mary	d_C	3
t_8	John	d_B	7

$hierarchy$

	Name	Boss	Depart
t_{20}	Adrian	Peter	d_A
t_{21}	Adrian	Peter	d_B
t_{22}	Adrian	John	d_B
t_{23}	John	John	d_B

$workers^{ext}$

	Name	Depart	SLevel
t_{31}	John	d_C	5
t_{32}	John	d_F	8

$hierarchy^{ext}$

	Name	Boss	Depart
t_{41}	Peter	John	d_B
t_{42}	Jay	Bill	d_C
t_{43}	Peter	Peter	d_A
t_{44}	Peter	Gates	d_A

Example 1 *Let $\mathbf{r}_{ex} = \{workers, hierarchy\}$ be the database instance of $\mathbf{R} = \{Workers(Name, Depart, SLevel), Hierarchy(Name, Boss, Depart)\}$ and $\mathbf{r}_{ex}^{ext} = \{workers^{ext}, hierarchy^{ext}\}$ the set of extra tuples that can be added to $\mathbf{r}_{ex}$ (Figure 1). For an easier reading, we represent each attribute name by its first letter. Moreover, let γ_1, γ_2, γ_3 and γ_4 be the following integrity constraints over $\mathbf{R}$:*

- *γ_1 is the DC $[Workers, \phi']$, where $\phi' \equiv (t_1[N, D] = t_2[N, D] \Rightarrow t_1[S] = t_2[S])$, i.e. every worker and department has a unique security level.*
- *γ_2 is the TGD $[Workers, \emptyset \sqsubseteq Hierarchy, \phi'']$, where $\phi'' \equiv (t_1[N, D] = t_2[N, D])$, i.e. for every worker there is a boss with respect to the same department.*
- *γ_3 is the DC $[(Workers, Hierarchy, Workers), \phi_3]$, where $\phi_3 = (t_1[N, D] = t_2[N, D] \wedge t_2[N] \neq t_2[B] \wedge t_2[B, D] = t_3[N, D]) \Rightarrow t_1[S] < t_3[S]$, i.e. every worker has a lower security level than each one of its bosses with respect to the same department.*
- *γ_4 is the DC $[(Hierarchy, Hierarchy), \phi_4]$, where $\phi_4 = (t_1[B] = t_2[B] \Rightarrow t_1[D] = t_2[D])$, i.e. every boss has an unique department.*

Let $\Sigma_{ex} = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4\}$. Then $\mathbf{r}_{ex}$ is inconsistent wrt Σ_{ex}, because, e.g., t_2 and t_3 do not satisfy γ_1.

2.2 Conflicting Sets

In order to restore the consistency of a database, we need to investigate the causes of the inconsistency. To this end, we find out tuples that are in conflict, i.e. that fail to satisfy one of the integrity constraints. We define four types of conflicts that allow to identify which inconsistencies must be *resolved* in order to restore the database consistency. In our setting, consistency restoration can only be achieved by deleting or inserting database tuples.

Definition 1 (d-conflicting set) *Let γ be an integrity constraint. A set of tuples $\mathbf{s} = \{t_1, ..., t_n\}$ is a **directly conflicting (d-conflicting)** set wrt γ if: (1) γ is a DC, $[(R_1, ..., R_n), \phi]$, and $\neg\phi(t_1, ..., t_n)$; or, (2) γ is a TGD, $[(R_1, ..., R_n), \phi_1 \sqsubseteq (R_{n+1}, ..., R_m), \phi_2]$, $\phi_1(t_1, ..., t_n)$ and $\mathbf{s} \not\models \gamma$. We say that the non-empty sets of tuples $\mathbf{s}'$ and $\mathbf{s}''$ are in **d-conflict** wrt γ if $\mathbf{s}' \cup \mathbf{s}''$ is a* d-conflicting *set wrt γ.*

A set of tuples $\mathbf{r}$ is inconsistent wrt Σ if only if it contains a *d-conflicting* set wrt some $\gamma \in \Sigma$. Hence, if we remove a tuple from all d-conflicting sets the resulting database is consistent [1]. A *d-conflicting* set may not be *minimum*, i.e., it may contain a proper *d-conflicting* subset. A **minimal** integrity constraint is an integrity constraint γ such that all d-conflicting sets wrt γ are minimum. In this work, we assume that our integrity constraints are *minimal*.

Example 2 *Examples of d-conflicting sets: (1) wrt γ_1, $\{t_2, t_3\}$ and $\{t_5, t_6\}$; (2) wrt γ_2, $\{t_1\}$, $\{t_6\}$ and $\{t_7\}$; (3) wrt γ_3, $\{t_1, t_{20}, t_2\}$; (4) wrt γ_4, $\{t_{20}, t_{21}\}$.*

We can also restore the consistency of a database by adding tuples. However, in contrast to the deletion of tuples, the addition of tuples may introduce new inconsistencies. For this matter we introduce the notion of *guarantor*.

Definition 2 (guarantor) *Let Σ be a set of integrity constraints. Given a d-conflicting set $\mathbf{s}$ wrt $\gamma \in \Sigma$, we say that a set of tuples $\mathbf{r}$ is a **guarantor** of $\mathbf{s}$ wrt Σ if $\mathbf{r}$ is a minimal set of tuples such that $\mathbf{s} \cup \mathbf{r} \models \Sigma$. The set that contains all the guarantors of $\mathbf{s}$ wrt Σ is represented by $Grt(\mathbf{s}, \Sigma)$.*

A guarantor of a set $\mathbf{s}$ is a non-empty set of tuples that $\mathbf{s}$ may *depend* on to be consistent. A guarantor is only related with TGDs because with respect to DCs conflicting sets there are no guarantors. A guarantor is *finite* if the given set of TGDs is acyclic.

Example 3 *$\{t_{20}\}$ is a guarantor of $\{t_1\}$ wrt γ_2. $Grt(\{t_2, t_3\}, \gamma_1) = Grt(\{t_1, t_{20}, t_2\}, \gamma_3) = Grt(\{t_{20}, t_{21}\}, \gamma_4) = \emptyset$.*

Given a database $\mathbf{r}$ and a set of integrity constraints Σ, $\mathbf{r}$ is consistent if only if $\mathbf{r}$ contains a guarantor for each *d-conflicting* set, in $\mathbf{r}$, wrt some $\gamma \in \Sigma$ [1].

In the following, we introduce an *indirect* conflict relation that can be created by a set of TGDs and a single integrity constraint.

Definition 3 (i-conflicting set) *Let $\Sigma = \Sigma_{tgd} \cup \{\gamma\}$ be a set of integrity constraints, where Σ_{tgd} is a set of TGDs. We say that $\mathbf{s} = \mathbf{s}_d \cup \mathbf{s}_i$ is an **i-conflicting set** (indirectly conflicting set) with respect to Σ_{tgd} and γ, if $\mathbf{s} \not\models \Sigma_{tgd}$, $\mathbf{s}_d \cap \mathbf{s}_i = \emptyset$ and there is a guarantor set $\mathbf{s}_w$ of $\mathbf{s}_d$ wrt Σ_{tgd}, where $\mathbf{s}_i$ is a minimum set of tuples that is in **d-conflict** with a subset of $\mathbf{s}_w$ wrt γ. We also say that the non-empty sets of tuples $\mathbf{s}'$ and $\mathbf{s}''$ are in i-conflict wrt Σ_{tgd} and γ if $\mathbf{s}' \cup \mathbf{s}''$ is an i-conflicting set wrt Σ_{tgd} and γ.*

An a *i-conflicting* set represents an indirect conflict relation between a set of tuples and a guarantor of another set of tuples. If a set of tuples r contains an *i-conflicting* set wrt Σ it does not entail that r is inconsistent wrt Σ. Moreover, an *i-conflicting* set could be non-minimum [1].

For notation purposes, given a set of integrity constraints Σ such that $\Sigma_{tgd}, \{\gamma\} \subseteq \Sigma$, we say that $\mathbf{s} \models \{\Sigma_{tgd} \triangleright \gamma\}$ if $\mathbf{s}$ is not an i-conflicting set wrt Σ_{tgd} and γ, where "$\Sigma_{tgd} \triangleright \gamma$" denotes a *derived constraint* from Σ. Moreover, we denote by Σ^* the union between Σ and the set of all *derived constraints* from Σ. We call each subset of Σ^* a set of *constraints*. Finally, we define $[\Sigma_{tgd} \triangleright \gamma]^+ = \Sigma_{tgd} \cup \{\gamma\}$ that is naturally extended for a set of constraints.

Example 4 *$\{t_1, t_{20}\}$ and $\{t_4, t_{20}\}$ are i-conflicting sets wrt $\{\gamma_2\} \triangleright \gamma_4$ because $\{t_{20}, t_{21}\}$ is a d-conflicting set wrt γ_4 and $\{t_{20}\}$ and $\{t_{21}\}$ are guarantors of $\{t_1\}$ and $\{t_4\}$ wrt γ_2, respectively.*

Given the conflict relations mentioned above, we say that a set is a **conflicting set** set wrt a set of integrity constraints Σ if it is a d-conflicting or an i-conflicting set wrt every $\gamma \in \Sigma$.

2.3 Repairs

Our work is based on the *S-Repair* model [3, 9] to restore the consistency of a database instance. This model assumes that the original database is neither consistent nor complete. Thus, the consistency restoration of a database is achieved by deleting or inserting tuples.

Definition 4 (S-Repair) *Given a database $\boldsymbol{r}$, a set $\boldsymbol{r}_e$ of extra tuples and a set of integrity constraints Σ, a database $\boldsymbol{r}'$ is a **repair** of $\boldsymbol{r}$ wrt Σ and $\boldsymbol{r}_e$ if $\boldsymbol{r}' \models \Sigma$ and $(\boldsymbol{r} - \boldsymbol{r}') \cup (\boldsymbol{r}' - \boldsymbol{r})$ is $minimal$ and $(\boldsymbol{r}' - \boldsymbol{r}) \subseteq \boldsymbol{r}_e$.*

Notice that if the set of extra tuples is empty or the set of integrity constraints only contains denial constraints, a repair is a maximal consistent subset of the original (inconsistent) database (*X-Repair* model, [3, 9]), because, in these cases, a repair is constructed by only deleting tuples. To simplify, we omit the set of integrity constraints and the set of extra tuples if they are known from the context.

Example 5 *$\boldsymbol{r}_{ex1} = \{t_1, t_3, t_4, t_5, t_8, t_{20}, t_{22}, t_{23}, t_{41}, t_{43}\}$, $\boldsymbol{r}_{ex2} = (\boldsymbol{r}_{ex1} - \{t_5\}) \cup \{t_6\}$, $\boldsymbol{r}_{ex3} = (\boldsymbol{r}_{ex1} - \{t_1, t_3\}) \cup \{t_2\}$ and $\boldsymbol{r}_{ex4} = (\boldsymbol{r}_{ex1} - \{t_1, t_{20}, t_{43}\}) \cup \{t_{21}, t_{44}\}$ are repairs of $\boldsymbol{r}_{ex}$ wrt Σ_{ex}, $\boldsymbol{r}_{ex}^{ext}$.*

2.4 Preferences

In order to establish preferences among repairs, we extend the notion of *priority* between pairs of conflicting tuples, introduced in [13], to a non-empty set of conflicting tuples. A priority can reflect, for example, the confidence of the accuracy placed by the user in distinct tuples that was propagated via data provenance analysis.

Definition 5 (Priority) *Let Σ be a set of constraints and $\boldsymbol{r}$ a database instance. A **priority** (wrt Σ) is a binary relation $\succ \subseteq \mathcal{P}(\boldsymbol{r}) \times \boldsymbol{r}$ only defined on conflicting sets of tuples, i.e. $\boldsymbol{s} \succ t$ if $\boldsymbol{s} \cup \{t\}$ is a conflicting set wrt Σ, and is acyclic, i.e. there does not exist $t \in \boldsymbol{r}$ and $\boldsymbol{s} \subseteq \boldsymbol{r}$ such that $t \in \boldsymbol{s}$ and $\boldsymbol{s} \succ^* t$, where $\succ^*$ is the transitive closure of $\succ$. If $\boldsymbol{s} \succ t$ we say that $\boldsymbol{s}$ dominates t wrt $\succ$.* [3]

Given that a priority is based on a preference, if $\mathbf{s}$ dominates t wrt $\succ$ then t is the *least element* of $\mathbf{s}' \cup \{t\}$ wrt $\succ$. Once again, for notation simplification purposes, we omit the set of integrity constraints from the priority symbol if it is known from the context.

Example 6 *Let $\succ_1 = \{(\{t_6\}, t_5)\}$, $\succ_2 = \{(\{t_2\}, t_3), (\{t_5\}, t_6), (\{t_2, t_{20}\}, t_1)\}$ and $\succ_3 = \{(\{t_{21}\}, t_1), (\{t_{21}\}, t_{43}), (\{t_{21}\}, t_{20})\}$ be priorities over $\boldsymbol{r}_{ex} \cup \boldsymbol{r}_{ex}^{ext}$ wrt Σ_{ex}. We have that $\{t_2\} \not\succ_1 t_5$, $\{t_6\} \succ_1 t_5$, $\{t_2\} \succ_2 t_3$, $\{t_2, t_{20}\} \succ_2 t_1$ and $\{t_{21}\} \succ_3 t_1$.*

In the following, we define a priority over sets of tuples with respect to an underlying set of tuples.

[3] $\succ^*$ is defined as follows: If $\mathbf{s} \succ t$ then $\mathbf{s} \succ^* t$; and, If $\mathbf{s} \succ t$ and $\mathbf{s}' \cup \{t\} \succ t'$ then $\mathbf{s} \succ t'$.

Definition 6 (Priority wrt a set of tuples) *Let $\boldsymbol{r}$ be a database instance and $\succ$ be a priority over $\boldsymbol{r}$. Given $\boldsymbol{s} \subseteq \boldsymbol{r}$, $\boldsymbol{r}_1 \subseteq \boldsymbol{s}$, $\boldsymbol{r}_2 \subseteq (\boldsymbol{r} - \boldsymbol{s})$ and $\boldsymbol{s}' = (\boldsymbol{s} - \boldsymbol{r}_1) \cup \boldsymbol{r}_2$, we write $\boldsymbol{r}_2 \succ^{\boldsymbol{s}} \boldsymbol{r}_1$ if*

$$\forall_{t \in \boldsymbol{r}_1} \quad \exists_{\boldsymbol{r}'_2 \subseteq \boldsymbol{s}' \wedge \boldsymbol{r}'_2 \cap \boldsymbol{r}_2 \neq \emptyset} \quad \boldsymbol{r}'_2 \succ t$$

Moreover, if $\boldsymbol{r}_2 \succ^{\boldsymbol{s}} \boldsymbol{r}_1$ we say that $\boldsymbol{r}_2$ dominates $\boldsymbol{r}_1$ wrt $\boldsymbol{s}$.

Example 7 *Let $\boldsymbol{s} = \{t_{20}\}$. $\{t_2\} \succ_2^{\boldsymbol{s}} \{t_1\}$, because $\{t_2, t_{20}\} \succ_2 \{t_1\}$; $\{t_{20}\} \not\succ_2^{\boldsymbol{s}} \{t_3\}$, because $\{t_{20}\} \not\succ_2 \{t_3\}$.*

[13] introduces three important notions of optimal repair that rely on a priority's preference information to establish preferences among repairs wrt functional dependencies. We extend those notions to our class of integrity constraints and repair model used.

Definition 7 (optimal repair) *Let $\boldsymbol{r}'$ be a repair of $\boldsymbol{r}$ wrt Σ, $\boldsymbol{r}_e$ a set of extra-tuples and $\succ$ a priority over $\boldsymbol{r}_t = \boldsymbol{r} \cup \boldsymbol{r}_e$ wrt Σ.*

- *$\boldsymbol{r}'$ is a **locally optimal** repair wrt $\succ$, if for no tuple $t_1 \in \boldsymbol{r}'$ there is a tuple $t_2 \in (\boldsymbol{r}_t - \boldsymbol{r}')$ such that $t_2 \succ^{\boldsymbol{r}'} t_1$ and $(\boldsymbol{r}' - \{t_1\}) \cup \{t_2\}$ is a consistent set wrt Σ.*
- *$\boldsymbol{r}'$ is a **semi-globally optimal** repair wrt $\succ$, if for no subset $\boldsymbol{s} \subseteq \boldsymbol{r}'$ there is a tuple $t \in (\boldsymbol{r}_t - \boldsymbol{r}')$ such that $t \succ^{\boldsymbol{r}'} \boldsymbol{s}$ and $(\boldsymbol{r}' - \boldsymbol{s}) \cup \{t\}$ is a consistent set wrt Σ.*
- *$\boldsymbol{r}'$ is a **globally optimal** repair wrt $\succ$, if for no subset $\boldsymbol{s} \subseteq \boldsymbol{r}'$ there is a set of tuples $\boldsymbol{s}' \subseteq (\boldsymbol{r}_t - \boldsymbol{r}')$ such that $\boldsymbol{s}' \succ^{\boldsymbol{r}'} \boldsymbol{s}$ and $(\boldsymbol{r}' - \boldsymbol{s}) \cup \boldsymbol{s}'$ is a consistent set wrt Σ.*

The notions of optimal repair can be roughly explained as follows: (1) A repair is *locally optimal* if it does not have a tuple that can be replaced by a conflicting tuple with a higher priority, such that the resulting set is consistent; (2) A repair is a *semi-globally optimal* if it does not have a subset of tuples that can replaced by a conflicting tuple with higher priority, such that the resulting set is consistent; (3) A repair is a *globally optimal* if it does not have a subset of tuples that can replaced by a set of tuples with a higher priority, such that the resulting set is consistent. Thus, a globally optimal repair is a semi-globally optimal repair and a semi-globally optimal repair is a locally optimal repair.

In our setting, a priority is defined over the original database instance and the set of extra tuples. Thus, we take into account not only the priorities among the tuples of the original database but also among the extra tuples. In [13] this situation was not considered. Notice, however, that an extra tuple can have a higher priority than a tuple from the database. As a result, each of the consistent resulting sets may not be a subset of a repair.

Example 8 *$\boldsymbol{r}_{ex1}$ is not a locally optimal repair wrt $\succ_1$ because $\{t_6\} \succ_1 t_5$ and $\boldsymbol{r}_{ex2}$ is consistent. $\boldsymbol{r}_{ex1}$ is a locally optimal repair wrt $\succ_2$, but it is not a semi-globally optimal repair wrt $\succ_2$, because $\{t_2\} \succ_2 t_3$, $\{t_2, t_{20}\} \succ_2 t_1$ and $\boldsymbol{r}_{ex3}$ is consistent. $\boldsymbol{r}_{ex1}$ is a semi-global optimal repair wrt $\succ_3$, but it is not globally optimal repair wrt $\succ_3$, because $\{t_{21}\} \succ_3 t_{43}$, $\{t_{21}\} \succ_3 t_{20}$, $\{t_{21}\} \succ_3 t_1$ and $\boldsymbol{r}_{ex4}$ is consistent. $\boldsymbol{r}_{ex1}$ is a global optimal repair wrt $\emptyset$.*

3 The Argumentation Framework

In this section we present our Argumentation Framework. We introduce the notions of argument, counter-argument and argumentation tree, which extends work of [12]. We assume that the set of integrity constraints is composed of *minimal* integrity constraints. In this way, we guarantee that every d-conflicting is minimal. We use A, B, ... to denote arguments.

3.1 Argument

An argument represents a statement about the relationship between a set of tuples and a set of integrity constraints by identifying the conflicting sets and consistent sets in a database. In this way, we are not only able to identify the causes of a database inconsistency but also to represent consistency restoration procedures (i.e. deletion and insertion of a tuple).

Definition 8 (argument) *Let Σ' be a set of integrity constraints and $\boldsymbol{r}$ and $\boldsymbol{s}$ be sets of tuples, where $\boldsymbol{r} \cap \boldsymbol{s} = \emptyset$, and $\emptyset \subset \Sigma \subseteq (\Sigma')^*$. An **argument** is a triple $\langle \boldsymbol{s}, \boldsymbol{r}, \Sigma \rangle$ such that :*

1. *If $\boldsymbol{r} = \emptyset$ then $\boldsymbol{s} \models \Sigma$.*
2. *If $\boldsymbol{r} \neq \emptyset$ then $\boldsymbol{s} \cup \boldsymbol{r}$ is a conflicting set wrt Σ.*
3. *If $\boldsymbol{r} \neq \emptyset$ and $\boldsymbol{s} \neq \emptyset$ then $\#\boldsymbol{r} = 1$.*

We say that $\langle \boldsymbol{s}, \boldsymbol{r}, \Sigma \rangle$ is an argument for Σ, that Σ is the consequent (conclusion) of the argument and that $\boldsymbol{s}$ and $\boldsymbol{r}$ are the positive *and* negative *support of the argument, respectively.*

For an argument $A = \langle \mathbf{s}, \mathbf{r}, \Sigma \rangle$, we define the functions $[A]_p = \mathbf{s}$, $[A]_n = \mathbf{r}$, $[A]_s = \mathbf{s} \cup \mathbf{r}$ and $[A]_{ic} = \Sigma$, which are naturally extended to sets of arguments.

An argument is composed by a support and a conclusion. The support is divided into a *positive* and *negative* part, each being a set of tuples. The conclusion is composed by a set of constraints.

An argument $\langle \mathbf{s}, \mathbf{r}, \Sigma \rangle$ represents the following statements: If $\mathbf{r} \neq \emptyset$ then both supports are in conflict wrt the conclusion. Otherwise, the support is consistent wrt the conclusion. An argument also represents a consistency restoration procedure: If $\mathbf{r} \neq \emptyset$, in order to satisfy Σ, a database $\mathbf{r}'$, such that $\mathbf{s} \subseteq \mathbf{r}'$, must verify that $\mathbf{r} \not\subseteq \mathbf{r}'$. That is, if we want to restore the consistency of our database and keep the tuples from the positive support we must delete the tuples from the negative support. However, the previous restoration procedure may not be unique when TGDs are considered, because we also can *resolve* a conflict by adding tuples.

Condition (1) of Definition 8 guarantees that if there is no negative support the positive support must satisfy the conclusion. Given this condition, we can use an argument to denote, for example, that a set of tuples (positive support) satisfies a set of constraints (conclusion). Condition (2) ensures that if there is a negative support then the union of both supports is a conflicting set wrt the conclusion. Condition (3) ensures that if both supports are non-empty then the negative support is composed of only one tuple, the *proposed* tuple to be deleted. Notice that the support of an argument could have irrelevant tuples, because it can be not minimum wrt conclusion.

Example 9 *Examples of arguments: $\langle \{t_1, t_2\}, \emptyset, \{\gamma_1\} \rangle$, $\langle \emptyset, \{t_6\}, \{\gamma_2\} \rangle$ and $\langle \{t_1, t_{20}\}, \{t_2\}, \{\gamma_3\} \rangle$. Examples of non-arguments: $\langle \{t_2, t_3\}, \emptyset, \{\gamma_1\} \rangle$ because it fails on the Condition (1); $\langle \{t_1\}, \{t_2\}, \{\gamma_1\} \rangle$ because it fails on the Condition (2); $\langle \{t_1\}, \{t_2, t_{20}\}, \{\gamma_1\} \rangle$ because it fails on the Condition (3).*

The following definition formalizes a degree of dependency between arguments, which is useful in the following subsections.

Definition 9 (more conservative) *Let $A = \langle \boldsymbol{s}, \boldsymbol{r}, \Sigma \rangle$ and $B = \langle \boldsymbol{s}', \boldsymbol{r}', \Sigma' \rangle$ be arguments. A is **more conservative** than B if $\Sigma \subseteq \Sigma'$ and $\boldsymbol{s} \subset \boldsymbol{s}'$ or $\boldsymbol{r} \subset \boldsymbol{r}'$.*

We say that argument A is a *maximal* conservative argument if there is no argument B that is more conservative than A.

Example 10 *$\langle \{t_1\}, \emptyset, \{\gamma_3\} \rangle$ is more conservative than $\langle \{t_1, t_7, t_{20}\}, \emptyset, \{\gamma_1, \gamma_3\} \rangle$.*

3.2 Counter-Argument

In order to refute a given argument we need to define the notion of *counter-argument*. In our setting, a counter-argument represents an alternative consistency restoration procedure for a given argument and can be one of two types: *undercut* or *rebuttal*.

Definition 10 (undercut) *An argument $A = \langle \boldsymbol{s}, \boldsymbol{r}, \Sigma \rangle$ is an **undercut** for an argument $B = \langle \boldsymbol{s}', \boldsymbol{r}', \Sigma' \rangle$, if $\emptyset \subset \boldsymbol{r} \subseteq \boldsymbol{s}'$ and (1) $\Sigma \not\subseteq \Sigma'$; or, (2) $\boldsymbol{r}' \neq \emptyset$ and $[A]_s \neq [B]_s$; or, (3) $\boldsymbol{r}' = \emptyset$ and $[A]_s \not\subseteq [B]_p$.*

An undercut of an argument A represents a argument that is in conflict with the support of A. An undercut also denotes an alternative consistency restoration procedure, in this case a tuple deletion, that can be applied instead of the procedure denoted by A.

Conditions (1,2,3) denote the different ways an argument can be a undercut: (1) By denoting a conflict wrt a different set of conclusion; or, (2) By having a different support set; or, (3) By having new tuples in the support. Conditions (2,3) prevent redundancy among arguments and respective undercuts by not allowing the construction of "symmetric" counter-arguments.

Example 11 *$A = \langle \{t_3\}, \{t_2\}, \{\gamma_1\} \rangle$ is an undercut for $B = \langle \{t_1, t_2\}, \emptyset, \{\gamma_1\} \rangle$. A is not an undercut for $C = \langle \{t_1, t_{20}\}, \{t_2\}, \{\gamma_3\} \rangle$, because $\{t_2\} \not\subseteq \{t_1, t_{20}\}$. $D = \langle \{t_1, t_2\}, \{t_{20}\}, \{\gamma_3\} \rangle$ is not an undercut for C. $E = \langle \emptyset, \{t_1\}, \{\gamma_2\} \rangle$ is an undercut for $F = \langle \{t_1, t_{20}\}, \emptyset, \{\gamma_2\} \rangle$.*

Definition 11 (rebuttal) *An argument $\langle \boldsymbol{s}', \emptyset, \Sigma' \rangle$ is a **rebuttal** for an argument $\langle \boldsymbol{s}, \boldsymbol{r}, \Sigma \rangle$ such that $\boldsymbol{r} \neq \emptyset$, if: (1) $\boldsymbol{s} \cup \boldsymbol{r} \subset \boldsymbol{s}'$; and, (2) $\Sigma^+ \subseteq \Sigma'$.*

A rebuttal of an argument A represents a *counter-argument* that given the support of A it contradicts its conclusion. An rebuttal also denotes an alternative consistency restoration procedure, in this case a tuple insertion, that can be applied instead of the procedure denoted by A (a tuple deletion). This situation happens when the set of constraints includes TGDs.

Conditions $(1, 2)$ assure that the rebuttal of an argumentation has a superset support and conclusion, i.e., it adds new tuples to the support of argument that allows the satisfaction of the conclusion.

Example 12 *$\langle \{t_1, t_{20}, t_{22}\}, \emptyset, \{\gamma_2 \triangleright \gamma_4, \gamma_2, \gamma_4\} \rangle$ is a rebuttal for $\langle \{t_{20}\}, \{t_1\}, \{\gamma_2 \triangleright \gamma_4\} \rangle$. $\langle \{t_3\}, \emptyset, \{\gamma_1\} \rangle$ is not a rebuttal for $\langle \{t_1, t_2, t_{20}\}, \emptyset, \{\gamma_3\} \rangle$.*

Definition 12 (sound/strict counter-argument) *Let Σ be a set of constraints, $\succ$ a priority wrt Σ and $\Sigma' \subseteq \Sigma$. An argument $A = \langle \boldsymbol{s}, \boldsymbol{r}, \Sigma' \rangle$ is a **sound** (resp. **strict**) **counter-argument** wrt $\succ$ for an argument B if: (1) A is a counter-argument for B; and, (2) $\neg \exists_{t \in \boldsymbol{s}} \quad \boldsymbol{r} \cup (\boldsymbol{s} - \{t\}) \succ t$ (resp. $\boldsymbol{s} \succ \boldsymbol{r}$) or $\boldsymbol{s} = \emptyset$ or $\boldsymbol{r} = \emptyset$.*

A *strict counter-argument* is an argument where the positive support dominates the negative support. A *sound counter-argument* is an argument that no tuple of the positive support is dominated by the others tuples of the argument support. Hence, a strict counter-argument is also sound.

Example 13 *$\langle \{t_2, t_{20}\}, \{t_1\}, \{\gamma_3\} \rangle$ is a sound counter-argument wrt $\succ_1$ and a strict counter-argument wrt $\succ_2$ for $\langle \{t_1\}, \{t_{21}\}, \{\gamma_2 \triangleright \gamma_4\} \rangle$, because $\{t_2, t_{20}\} \succ_2 \{t_1\}$. $\langle \{t_6\}, \{t_5\}, \{\gamma_1\} \rangle$ is not strict counter-argument wrt $\succ_2$ for $\langle \{t_1, t_5, t_{22}\}, \emptyset, \{\gamma_3\} \rangle$, because $\{t_6\} \not\succ_2 t_5$. $\langle \{t_5\}, \{t_6\}, \{\gamma_1\} \rangle$ is a non sound counter-argument wrt $\succ_1$ for $\langle \{t_6\}, \emptyset, \{\gamma_1\} \rangle$, because $\{t_6\} \succ_1 t_5$.*

Definition 13 (canonical counter-argument) *Let Σ be a set of constraints, A a counter-argument of argument B such that $[A]_{ic} \subseteq \Sigma$. We say that A is a **canonical counter-argument** wrt Σ for an argument B if:*

1. *if $[A]_p \neq \emptyset$ then (a) A is a maximal conservative counter-argument for B; and (b) $[A]_s \not\subseteq [B]_p$; and,*
2. *if $[A]_p = \emptyset$ then there is no counter-argument C of B such that $[C]_p = \emptyset$, $[A]_n \subset [C]_n$ and $[A]_{ic} \subseteq [C]_{ic}$; and,*
3. *there is no argument D such that $[A]_{ic} \subset [D]_{ic} \subseteq \Sigma$ and obeys (1) and (2).*

A canonical counter-argument represents a possible infinite set of "equivalent" counter-arguments for a given argument. Condition (1a) ensures that the support of a counter-argument is free from redundant tuples. This condition is required because the support of an argument can be non-minimum. Condition (1b) ensures that if the support of a rebuttal A for an argument B is a subset of $[B]_p$ then it must have an empty positive support. Conditions (1b) and (2) ensure that the canonical counter-argument for B represents the "strongest" counter-argument for B because it has an empty positive support and, therefore, strict wrt any priority. Finally, Condition (3) ensures that the canonical argument is the most conclusive possible by checking that it has a maximal set of constraints.

Example 14 *$\langle\{t_6\},\{t_5\},\{\gamma_1\}\rangle$ is a strict canonical counter-argument wrt $\succ_1$ for $\langle\{t_1,t_5,t_{22}\},\emptyset,\{\gamma_3\}\rangle$. $\langle\{t_5\},\{t_6\},\{\gamma_1\}\rangle$ is not a canonical counter-argument for $\langle\{t_1,t_5,t_6\},\emptyset,\{\gamma_3\}\rangle$, because it fails Condition (1a). $\langle\emptyset,\{t_5,t_6\},\{\gamma_1\}\rangle$ is a canonical counter-argument for $\langle\{t_1,t_5,t_6\},\emptyset,\{\gamma_3\}\rangle$. $\langle\{t_1,t_{20},t_{22}\},\emptyset,\{\gamma_2,\gamma_4,\gamma_2 \triangleright \gamma_4\}\rangle$ is not a canonical counter-argument for $\langle\{t_{20}\},\{t_1\},\{\gamma_2 \triangleright \gamma_4\}\rangle$, because $\langle\{t_1,t_{20},t_{22}\},\emptyset,\{\gamma_1,\gamma_2,\gamma_4,\gamma_2 \triangleright \gamma_4\}\rangle$ is also a counter-argument and, therefore, it fails Condition (3).*

The following proposition shows that, if the set of integrity constraints is acyclic, we can obtain all canonical counter-arguments for argument in polynomial time wrt the size of the database. Hence, this framework presents a reasonable way to represent the alternative consistency reparations procedures without *extra* complexity.

Proposition 1 *Let Σ be a set integrity constraints, $\boldsymbol{r}$ a set of tuples and A an argument. If Σ is acyclic, finding all canonical counter-arguments for A wrt Σ and $\boldsymbol{r}$ can be done in polynomial time wrt the size of $\boldsymbol{r}$.*

3.3 Argumentation Tree

An argumentation tree represents, in a recursive form, the different ways an argument can be challenged, as its counter-arguments.

Definition 14 (argumentation tree) *Let Σ be a set of integrity constraints and $\boldsymbol{r}$ a set of tuples. An **argumentation tree** for A wrt $\boldsymbol{r}$ and Σ is a tree where nodes are arguments such that:*

1. *The root is the argument $A = \langle \boldsymbol{s}', \emptyset, \Sigma'\rangle$ such that $\boldsymbol{s}' \subseteq \boldsymbol{r}$ and $\Sigma' \subseteq \Sigma^*$; and,*
2. *There is no node B with an indirect ancestor node C such that $[B]_s \subseteq [C]_s \subseteq \boldsymbol{r}$ and $[B]_{ic} \subseteq [C]_{ic} \subseteq \Sigma^*$; and,*
3. *The children nodes of a node A' consist of canonical counter-arguments for A' that obey (2).*

Each argument of an argumentation tree is constructed based on a given set of tuples and set of integrity constraints. The support of each argument is a subset of the given set of tuples, and its conclusion a the set of constraints. Condition (2) ensures that an argumentation tree does not have cycles and Condition (3) ensures that each counter-argument is canonical. Given these conditions, we get the following result about the size of argumentation tree.

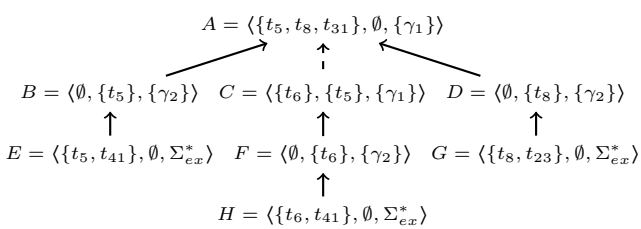
Proposition 2 *Let Σ be a set of integrity constraints, $\boldsymbol{r}$ a database instance and $\boldsymbol{r}^e$ a set of extra tuples. If the sets $\boldsymbol{r}^e$ and Σ are finite then all argumentation trees wrt $\boldsymbol{r} \cup \boldsymbol{r}^e$ and Σ are finite.*

For an argumentation tree $\mathfrak{T}$, each argument in $\mathfrak{T}$ is either an **attacking argument** or a **defensive argument**. The root of the tree is a defensive argument. If A_i is a defensive argument (resp. attacking), then any child of A_i is an attacking argument (resp. defensive). The set of defensive arguments is given by $D(\mathfrak{T})$ and the set of attacking arguments is given by $A(\mathfrak{T})$. Moreover, the set of attacking (resp. defensive) ancestor arguments of an argument A wrt $\mathfrak{T}$ is given by $\mathcal{A}(A,\mathfrak{T})$ (resp. $\mathcal{D}(A,\mathfrak{T})$). Finally, $At(\mathfrak{T})$ and $Dt(\mathfrak{T})$ denote the set of tuples that are first introduced in the tree $\mathfrak{T}$ uniquely by attacking and defensive arguments, respectively.

Definition 15 (full/single strict argumentation tree) *Let $\mathfrak{T}$ be an argumentation tree and $\succ$ a priority. $\mathfrak{T}$ is a **full argumentation tree** if the children nodes of any node A consist of all canonical counter-arguments for A. $\mathfrak{T}$ is a **single strict attacking** argumentation tree wrt $\succ$ if the children nodes of an attacking (resp. defensive) argument A, consist of sound (resp. at most, one strict) canonical counter-arguments for A wrt $\succ$.*

In a full argumentation tree, for each argument, all the canonical counter-arguments are taken into account. A single strict attacking argumentation tree is composed of sound defensive arguments with, at most, one strict attacking counter-argument. A full argumentation tree can be used to check the consistency of a set tuples [1].

Figure 2. The full argumentation tree $\mathfrak{T}_1$.



Definition 16 (successful argumentation tree) *We say that $\mathfrak{T}$ is **successful** if the leaf of every branch is a defensive argument. Moreover, say that $\mathfrak{T}$ is **full unsuccessful** if the leaf of every branch is an attacking argument.*

An argumentation tree is successful if all attacking arguments are defeated. On the other hand, it is full unsuccessful if all defensive arguments are defeated.

Example 15 *Figure 2 illustrates the full argumentation tree $\mathfrak{T}_1$ for the argument $A = \langle\{t_5,t_8,t_{31}\},\emptyset,\{\gamma_1\}\rangle$ wrt Σ_{ex} and $\boldsymbol{r}_{ex} \cup \boldsymbol{r}^{ext}_{ex}$. A dashed arrow represents a non-sound canonical argument wrt $\succ_2$ and a solid arrow represents a strict canonical argument wrt $\succ_2$. Moreover, $D(\mathfrak{T}_1) = \{A,E,F,G\}$, $A(\mathfrak{T}_1) =$*

$\{B, C, D, H\}$, $\mathcal{A}(H, \mathfrak{T}_1) = \{C\}$, $\mathcal{D}(H, \mathfrak{T}_1) = \{A, G\}$, $Dt(\mathfrak{T}) = \{t_5, t_8, t_{23}, t_{31}\}$ *and* $At(\mathfrak{T}) = \{t_6\}$. *Notice that* $t_{41} \notin Dt(\mathfrak{T}) \cup At(\mathfrak{T})$ *because* t_{41} *is introduced by a defensive and an attacking argument. The left and the right branches of the argumentation tree* $\mathfrak{T}_1$ *correspond to single strict attacking argumentation trees* $\mathfrak{T}_2$ *and* $\mathfrak{T}_3$*, respectively, for* A *wrt* Σ_{ex}, $\boldsymbol{r}_{ex} \cup \boldsymbol{r}_{ex}^{ext}$ *and* $\succ_2$. $\mathfrak{T}_1$ *is an unsuccessful tree, but not a full unsuccessful tree.* $\mathfrak{T}_2$ *and* $\mathfrak{T}_3$ *are successful trees.*

4 Optimal Repair Checking

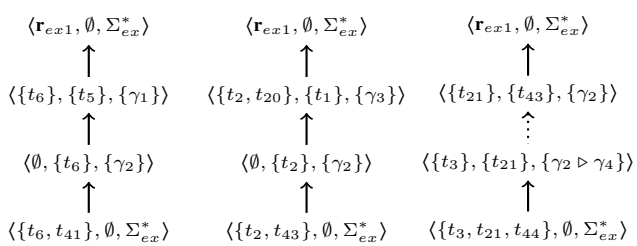
The following result shows how the optimality of a repair (local, semi-global and global) can be checked using our argumentation framework. These result also shows that our framework successfully captures all the conflicts among data and the possible consistency reparations procedures.

Theorem 1 (optimal repair checking) *Let* Σ *be a set of integrity constraint,* $\succ$ *a priority,* $\boldsymbol{r}$ *an inconsistent database wrt* Σ, $\boldsymbol{r}^{ext}$ *is a set of extra-tuples,* $\boldsymbol{r}_r$ *a repair of* $\boldsymbol{r}$ *wrt* $\boldsymbol{r} \cup \boldsymbol{r}^{ext}$ *and* Σ. *There is no full unsuccessful single strict attacking argumentation tree* $\mathfrak{T}$ *for* $\langle \boldsymbol{r}_r, \emptyset, \Sigma^* \rangle$ *wrt* Σ, $\boldsymbol{r}_{ex} \cup \boldsymbol{r}_{ex}^{ext}$ *and* $\succ$, *where* $\boldsymbol{s}' = [A(\mathfrak{T})]_p - \boldsymbol{r}_r$ *and* $\boldsymbol{s} = [A(\mathfrak{T})]_n$, *such that the children nodes of each attacking (resp. defensive) node of* $\mathfrak{T}$ *consist of all (resp. one, if exists) canonical counter-arguments such that:* (1) $Dt(\mathfrak{T}) = \boldsymbol{r}_r$; *and,* (2) $[A(\mathfrak{T})]_n \subseteq \boldsymbol{r}_r - [A(\mathfrak{T})]_p$; *and,* (3) $\forall_{B \in D(\mathfrak{T})} \forall_{A \in \mathcal{D}(D, \mathfrak{T})} \quad [B]_p \cap [A]_n = \emptyset$; *and*

(a) $\#\boldsymbol{s} = \#\boldsymbol{s}' = 1$ *and* $(\boldsymbol{r}_r - \boldsymbol{s}) \cup \boldsymbol{s}' \models \Sigma$ *iff* $\boldsymbol{r}_r$ *is a locally optimal repair wrt* Σ, $\succ$.
(b) $\#\boldsymbol{s}' = 1$ *iff* $\boldsymbol{r}_r$ *is a semi-globally optimal repair wrt* Σ, $\succ$.
(c) $\#\boldsymbol{s}' \geq 1$ *iff* $\boldsymbol{r}_r$ *is a globally optimal repair wrt* Σ, $\succ$.

Theorem 1 shows that there is an equivalence between a repair being locally, semi-globally and globally optimal and the existence of a single strict attacking argumentation tree. Condition (1) assures that defensive arguments do not introduce new tuples in the argumentation tree and, thus, are restricted to the tuples from the given repair or tuples introduced by attacking arguments. Condition (2) assures that attacking arguments do not have common tuples in the positive and negative support. This condition does not allow, e.g., an attacking argument to have in the positive support a tuple that was already *attacked*. Condition (3) assures that defensive arguments do not have tuples in the positive support that were already attacked in the branch. (a), (b) and (c) establish the equivalence wrt all optimal repair types.

Figure 3. The single strict attacking argumentation trees $\mathfrak{T}_4$, $\mathfrak{T}_5$ and $\mathfrak{T}_6$.



Example 16 *Figure 3 illustrates the single strict attacking argumentation trees* $\mathfrak{T}_4$, $\mathfrak{T}_5$ *and* $\mathfrak{T}_6$ *wrt* $\succ_1$, $\succ_2$ *and* $\succ_3$, *respectively, and* Σ_{ex} *for* $\langle \boldsymbol{r}_{ex1}, \emptyset, \Sigma_{ex}^* \rangle$. *Given* $\mathfrak{T}_4$ *we conclude, by Theorem 1, that* $\boldsymbol{r}_{ex1}$ *is not a locally optimal repair of* $\boldsymbol{r}_{ex}$ *wrt* $\succ_1$, *because the result of replacing* t_5 *by* t_6 *(i.e.,*$[A(\mathfrak{T}_4)]_p - \boldsymbol{r}_{ex1}$*) in* $\boldsymbol{r}_{ex1}$ *(i.e.,* $\boldsymbol{r}_{ex2}$*) is consistent wrt* Σ_{ex}. *Given* $\mathfrak{T}_5$, *we conclude, by Theorem 1, that* $\boldsymbol{r}_{ex1}$ *is not a semi-global optimal repair of* $\boldsymbol{r}_{ex}$ *wrt* $\succ_2$, *because, e.g., the result of replacing* $\{t_1, t_3\}$ *by* t_2 *(i.e.,*$[A(\mathfrak{T}_5)]_p - \boldsymbol{r}_{ex1}$*) in* $\boldsymbol{r}_{ex1}$ *(i.e.,* $\boldsymbol{r}_{ex3}$*) is consistent wrt* Σ_{ex}. *Given* $\mathfrak{T}_6$, *we conclude, by Theorem 1, that* $\boldsymbol{r}_{ex1}$ *is not a global optimal repair of* $\boldsymbol{r}_{ex}$ *wrt* $\succ_3$, *because the result, e.g., of replacing* $\{t_1, t_{20}, t_{43}\}$ *by* $\{t_{21}, t_{44}\}$ *(i.e.,*$[A(\mathfrak{T}_6)]_p - \boldsymbol{r}_{ex1}$*) in* $\boldsymbol{r}_{ex1}$ *(i.e.,* $\boldsymbol{r}_{ex4}$*) is consistent wrt* Σ_{ex}.

5 Discussion, Conclusions and Future Work

The aim of this paper has been to use argumentation to provide a clear and comprehensive understanding of the database repair process. We presented an argumentation framework that provides a way to identify and represent the conflicts between tuples but also the possible consistency restoration procedures that can be applied in each case. Proposition 1 shows that this is done without compromising its practicability. We extended the notions of locally, semi-globally and globally optimal repair [13] to show that we can also use this argumentation framework to check important repair related properties (Theorem 1). Our work extends the argumentation framework introduced in [12] to a broader and more common set of integrity constraints, namely, Denial Constraints and Tuple-Generating Dependencies classes. It also provides an improved alternative for the *conflict hypergraph* [8] that is used to succinctly represent the conflicts between data with respect to the Denial Constraints class. Although database repair is closely related to *Belief Revision*, besides [12], to our best of knowledge there is no work that proposed an argumentation-based approach to enhance the data consistency restoration process. Thus, we advocate that this work opens a new research line that embraces the Data Cleaning and Argumentation research areas.

As ongoing research, we propose to build a decision support system, based on our argumentation framework, that assists the user in the consistency restoration process by identifying the conflicts and suggesting possible repairing procedures. As future work, we propose to extend our framework to consider database user annotations [10] in the consistency restoration process.

REFERENCES

[1] E. Santos, 'An argumentation-based approach to database repair'. Unpublished. http://web.ist.utl.pt/esantos/tech-2010.pdf, 2010.
[2] L. Amgoud and C. Cayrol, 'A reasoning model based on the production of acceptable arguments', *Annals of Mathematics and Artificial Intelligence*, **34**(1-3), 197–215, 2002.
[3] M. Arenas, L. Bertossi, and J. Chomicki, 'Consistent query answers in inconsistent databases', in *PODS*, pp. 68–79. ACM Press, 1999.
[4] P. Atzeni and V. De Antonellis, *Relational database theory*, Benjamin-Cummings Publishing Co., Inc., 1993.
[5] C. Batini and M. Scannapieco, *Data Quality: Concepts, Methodologies and Techniques*, Data-Centric Systems and Apps., Springer, 2006.
[6] P. Besnard and A. Hunter, *Elements of Argumentation*, MIT Press,2008.
[7] C. Chesñevar, A. Maguitman, and R. Loui, 'Logical models of argument', *ACM Comput. Surv.*, **32**(4), 337–383, 2000.
[8] J. Chomicki and J. Marcinkowski, 'On the computational complexity of minimal-change integrity maintenance in relational databases', in *Inconsistency Tolerance*, vol. 3300 of *LNCS*, pp. 119–150.Springer, 2005.
[9] W. Fan, 'Dependencies revisited for improving data quality', in *PODS*, eds., M. Lenzerini and D. Lembo, pp. 159–170. ACM, 2008.
[10] W. Gatterbauer, M. Balazinska and and D. Suciu, 'Believe it or not: Adding belief annotations to databases', *PVLDB*, **2**(1), 1–12, 2009.
[11] E. Santos and J. P. Martins, 'A default logic based framework for argumentation', in *ECAI*, vol. 178 of *FAIA*, pp. 859–860. IOS Press, 2008.
[12] E. Santos and J. P. Martins, 'An argumentation framework for optimal repair checking', in *ICCP*, pp. 19 – 26, 2009.
[13] S. Staworko and J. Chomicki, 'Priority-based conflict resolution in inconsistent relational databases', *CoRR*, 2005.