Using Argumentation to Support the User Involvement In Data Cleaning

Emanuel Santos
INESC-ID and Technical University of Lisbon
esantos@ist.utl.pt

Helena Galhardas
INESC-ID and Technical University of Lisbon
hig@inesc-id.pt

ABSTRACT

Data cleaning processes are usually modeled as graphs of data transformations. The involvement of the users is important to manually correct data items that cannot be treated automatically. However, in the context of an iterative data cleaning process and multiple user involvement, the users feedback can be contradictory. In this paper, we propose the use of *Argumentation* to support the user involvement by determining which user data corrections should be applied during the data cleaning process. First, we introduce the notion of *based manual data repair instance* to represent the way users can provide the feedback required to manually clean data items and their corresponding justification. Second, we construct an argumentation framework and use argumentation acceptability criteria to determine which based manual data repair instances should be applied during the data cleaning process based on their consistency and reliability.

1. INTRODUCTION

Data cleaning processes are usually modeled as graphs of data transformations. These graphs typically involve a large number of data transformations, and must handle large amounts of data. The data transformations incorporate cleaning criteria that must be satisfied by a data flow graph. In general, it is not easy to conceive a graph of data transformations that produces always accurate data, i.e. data that satisfy the corresponding cleaning criteria. This happens because it is difficult to write data transformations that address all instances of data cleaning problems and, hence, it is not always possible to get a fully automated data cleaning solution. Thus, some of the output data of a data transformation may be rejected, because it cannot be automatically handled by the transformations.

In order to handle the rejected data, the involvement of the users responsible for executing the corresponding programs over real data is needed to tune data transformations. However, the tuning of data transformations can be insufficient to handle all the rejected data. In these cases, the user involvement is also needed to manually correct data items that cannot be treated automatically. However, the users feedback can be contradictory.

In this work, we propose the use of *Argumentation* [2] to support the user involvement in data cleaning processes. We propose to extend the notion of data cleaning graph by including (i) *based manual data repair instances*, *bmi* for short, to represent the way users can provide the feedback required to manually clean data items and their corresponding justification, and (ii) an argumentation framework to help determine which sets of based manual data repair instances should be applied during the execution of the data cleaning graph.

This paper is organized as follows. Section 2 describes our motivating example for our proposal. In Section 3 we recall the notion of Data Cleaning Graph and its operational semantics. In Section 4 we identify and address the inconsistency problem among manual data repairs instances. In Section 5 we introduce the notion of based manual data repair instance. In Section 6 we introduce an argumentation framework to determined which based manual data repair instances should be applied in the data cleaning graph. In Section 7 we discuss the related work and in Section 8 we summarize the conclusions and future work. Due to space limitations, all proofs are omitted. An extended version of this paper can be found in [14].

2. A MOTIVATING EXAMPLE

Our motivating example is based on a Portuguese Maritime Transport Registry Entity (PMTRE, for short) database that stores the portuguese maritime transport registry, such as boat owners, location, etc. The current version of this database has 387 tables. The largest table has more than 1,000,000 tuples. This table, named Entities, store data about individual and collective entities that own a registered boat. A simplification of the Entities table is illustrated in Figure 1, where eid stores the entity unique identification; name stores the entity name; ssn stores the entity social security number; date stores the birthday date of individual entities; type stores the entity's type, i.e., individual or collective, represented by "IND" and "COL" values, respectively; address stores the entity full address; and, finally, office stores the PMTRE office where the entity is registered. Some integrity constraints are associated to the Entities table schema. For instance, individual and collective entities have ssn values starting by 1 and 5, respectively.

The Entities table is manually maintained by PMTRE employees when a boat registration is inserted, updated or deleted. Several data quality problems in this table have been identified. For instance, non-standardized names (e.g., "JOSE VIEIRA ANDRADE" and "José Vieira Andrade") and dates (e.g., "Maio de 74"), duplicates (e.g., tuples 3 and 4) and inconsistent type and ssn values (e.g., tuple 3 has a type "IND" and a ssn number that starts by a 5). These particular examples can be automatically identified and cleaned. However, there are cases we believe only domain expert users are capable to successfully clean. For instance, some entity names are incomplete (e.g., tuple 7 with name "josé v. a.") and can only be completed by users that are able to identify and update them to their full name. Moreover, state-of-art procedures to identify duplicates, that are usually based on the use of approximate string matching, may not be enough to automatically identify duplicates. For example, tuples 2 and 7 are duplicates with very distinct names, "Vieira ANDRÉ" and "josé v. a.", respectively, and, thus, they may not be automatically identified as duplicates.

3. DATA CLEANING GRAPH

The previous examples shows the importance of involving the user in the data cleaning process to manually correct the data quality problems that cannot be handled automatically. For this purpose, [10] presents a modelling primitive named *Data Cleaning Graph* (DCG, for short) that supports the specification of a data cleaning process. A DCG is modelled as an acyclic graph of data transformations. Each of these data transformations accepts relations as input and produces one relation as output. The output relations can be explicitly expressed and associated with *quality constraints*. A quality constraint expresses criteria that the output of a data transformation must satisfy. Its purpose is to call the user's attention, i.e. developers and non-developers, for data quality problems existing in a relation that were not handled automatically by the transformation that produced it. To help users to identify where the problem is, the tuples responsible for the violation of a quality constraint are identified. These tuples are called *blamed tuples*. In order to include the user involvement, a DCG incorporates user feedback in the form of a *manual data repair* (*mdr*, for short) . A *mdr* represents an update, a insert or a delete action that can be performed by the users over a relation of the DCG. Each data repair action is defined over an updatable view, which, typically, is defined over the set of blamed tuples of corresponding relation. These views filter the information available in the relation and exclude non relevant information that users do not need to process in order to decide which are the appropriate data repair actions to apply. The data repair actions that users have decided to apply wrt a *mdr* are named *manual data repair instances* (*mis*, for short). These instances are applied during the execution of the DCG. [10] does not address their application order.

Typically, a data cleaning process begins with the development and tuning of data transformations, by the developers, until a reasonable number of tuples are properly cleaned automatically. The data cleaning process is an iterative process that may incorporate the user's feedback. Each iteration corresponds to an execution step of a DCG, which incorporates a sequence of tasks: (1) the computation of the applicable *mis* wrt the input tables of the DCG, and their application over the corresponding tuples. (2) the computation of the *blamed tuples* wrt the input tables of the DCG. (3) the selection of the data transformations that will be executed. The execution of the DCG can be partial or total and is performed by the developer or automatically; and, finally, (4) the execution of the selected data transformations. Their order of execution is determined by their dependency order defined in the DCG. The execution of each data transformation is followed by a sequence of tasks analogous to (1) and (2).

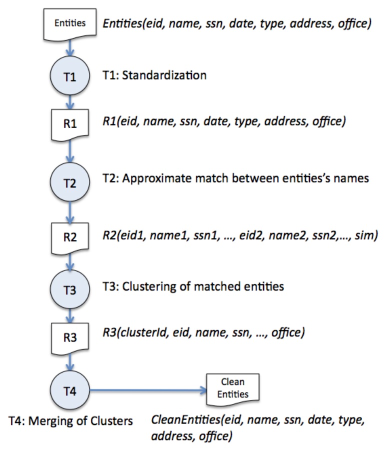

Entities
Entities(eid, name, ssn, date, type, address, office)
T1
T1: Standardization
R1
R1(eid, name, ssn, date, type, address, office)
T2
T2: Approximate match between entities's names
R2
R2(eid1, name1, ssn1, ..., eid2, name2, ssn2,..., sim)
T3
T3: Clustering of matched entities
R3
R3(clusterId, eid, name, ssn, ..., office)
T4
Clean Entities
T4: Merging of Clusters
CleanEntities(eid, name, ssn, date, type, address, office)

Figure 2: Excerpt of the DCG for Entities table.

Each execution step is preceded by the addition of *mis* by the users. Notice that, the result of each execution step relies on the results of the previous execution steps. For instance, if a tuple is updated by a *mi* in an execution step then the updated resulting tuple will appear in the following execution step, instead of the pre-updated tuple.

Figure 2 shows an excerpt of the DCG that was conceived for cleaning the Entities table so that the output table, CleanEntities, stores entity records with correct and standardized attribute values and no duplicates. In this cleaning graph, data transformation T1 produces the R1 table. Figure 4 shows an excerpt of instance of R1 that results from standardizing the attributes values of every tuple of the Entities table. The standardization consists of normalizing the name, address and office values, rewriting them so that a capital letter is used as the first letter and lower-case letters in the remaining letters of each word of these strings. Moreover, the date values are also standardized to the format "DD/MM/YYYY". T1 also transforms the type attribute according to the corresponding ssn attribute value. The duplicate elimination process is performed by the sequence of data transformations T2, T3 and T4. Data transformation T2 matches pairs of standardized Entities tuples by computing the Levenshtein distance between the name string values and storing it in the sim attribute of table R2. Data transformation T3 clusters the pairs of entity tuples with similarity value equal or higher than 0.8, i.e. $\mathsf{sim} \geq 0.8$, according to the transitive property and stores them in R3. Finally, data transformation T4 tries to merge each cluster produced by T3 into a single tuple and stores them in CleanEntities, which represents the cleaned data produced. The name and address values are merged by selecting the most complete name and address values in each cluster, respectively. The remaining attribute values are merged by selecting the most frequent values in each cluster.

Some quality constraints and *mdrs* we introduced in the graph. For the sake of simplicity and due to space limita-

eid	name	ssn	date	type	address	office
1	JOSE VIEIRA ANDRADE	150691111		IND	Loures, Lisbon	LISBON
2	Vieira ANDRÉ			IND	Praca da Liberdade, no 6	Porto
3	JACINTO MADEIRO	500726666		IND	Sines	Lisboa
4	Lda JACINTO MADEIRO			COL	Sines	Lisboa
5	José Vieira Andrade	150691111	1/7/2079	IND	Lisbon	Lisbon
6	a.			COL	Nazaré	Leiria
7	josé v. a.		Maio de 74	IND	Loures	PORTO

Figure 1: Entities table.

Mdr	Node	User actions	View
Mdr1	R1	delete update name, ..., office	Select name, ..., office From blamed(Qc1)

Figure 3: A manual data repair of the DCG.

eid	name	ssn	date	type	address	office
2	Vieira André			IND	Praça da Liberdade, 6	Porto
6	A.			COL	Nazaré	Leiria
7	José V. A.		5/1974	IND	Loures	Porto

Figure 4: An excerpt of R1 table.

tions, in the remaining of the paper we focus on table R1. Quality constraints are added to the DCG to call the users' attention for quality problems in the data produced by the transformations. The output of T1, R1, is subject to the quality constraint, Qc1: R1.address contains R1.office and fullName(R1.name) and NotNull(ssn) that denotes that: the entity must live in the same location of the PMTRE office, where the entity is registered, or, in other words the office string value must be contained in the address string value; the entity's name must be a full name, i.e. a string without acronyms or abbreviations; and, the entity's ssn must be not null. For instance, tuple 2 of the R1 table violates Qc1 because it has a null ssn value. Moreover, tuple 7 violates Qc1 because it has a non full name, a null ssn value, and an address value, "Loures", which does not include its office value, "Porto".

Manual data repairs were added to the DCG to incorporate the user's feedback in order to correct the identified data quality problems. Notice that, in this example, the feedback may not be only produced by the developers but also by "end-users" and/or "domain-experts", i.e, PMTRE employees. Thus, this data cleaning process allows the involvement of *multiple-users*, where different users with different degrees of expertise may add their feedback to the data cleaning process.

Figure 3 illustrates the *mdr* Mdr1 that is associated to the R1 table to involve the user in the correction of the blamed tuples. Mdr1 consists of a possible delete or update action. These actions can be applied to the tuples produced by the view projecting all attributes except the id attribute of the set of tuples blamed for the violation of Qc1, denoted by blamed(Qc1). For instance, given Mdr1, suppose that three users, named $U1$, $U2$ and $U3$ want to correct tuple 7 of Figure 4, which is a blamed tuple of Qc1 and, hence, belongs to the set tuples returned by the view associated with Mdr1. In order to correct tuple 7, users propose the following actions through Mdr1: $U1$ and $U2$ propose to update the office value of tuple 7 to "Loures" and "Lisboa", respectively, because they believe that if an entity is individual and its address is "Loures" then its office must be "Loures" and "Lisboa", respectively. $U3$ proposes to update the address value of tuple 7 to "Porto" because $U3$ believes that if an entity office is "Porto" then its address must be "Porto". These proposed updates of tuple 7 of R1 are represented by *mis* of Mdr1 and are illustrated in Figure 5 as mi1, mi2, mi3, respectively. In Figure 5 **Mdr** corresponds to the associated *mdr*; **tuple** corresponds to the key of the targeted tuple of R1; **value** denotes the attribute and the new updated value if the action is an update; or, otherwise, denotes the action that will be applied. In this simplified notation, for instance, office="Loures" denotes that the office attribute value is "Loures". Figure 5 also illustrates a *mi* mi4 added by a user that represents the deletion of tuple 6 of R1, which is also a blamed tuple of Qc1. In our running example, we assume that these *mis* were added by the users in the first execution step of the data cleaning process.

Mdr instance	Mdr	tuple	value
mi1	Mdr1	eid="7"	office="Loures"
mi2	Mdr1	eid="7"	office="Lisboa"
mi3	Mdr1	eid="7"	address="Porto"
mi4	Mdr1	eid="6"	delete

Figure 5: Some *mis* of the DCG.

One of the purposes of adding *mis* is to correct blamed tuples in such way that they satisfy the corresponding quality constraints. However, *mis* can also be added, for instance, (1) to correct non-blamed tuples; or, (2) to correct blamed tuples in such way that they do not resolve the violation of the corresponding quality constraints. For instance, an user may believe that the correct values of the tuple do not satisfy the corresponding quality constraint.

We identify three main problems related with the notion of *mdr* introduced in [10]: (1) Since the application of a single *mi* may not be enough to correct a tuple, the order of application of *mis* is crucial to determine the resulting tuples and, hence, the satisfaction of the corresponding quality constraints. However, [10] does not define a proper order of application of the *mis*; (2) In a *multiple-user* involvement setting, it is possible to gather user feedback, through *mis*, that may be conflicting. That is, distinct users may propose to apply opposite or incompatible *mis* to the same tuple. Hence, *conflicting mis* should not be applied together in the DCG. For instance, mi1 and mi2 represent two *conflicting mis* because they propose to update the office attribute to two distinct values, "Loures" and "Lisboa", respectively. Moreover, mi1 and mi3 can also be considered *conflicting* because mi1 updated value was decided based on the tuple's address value "Loures" which is in contradiction with mi2's address updated value, "Porto". [10] does not address the *multiple-user* involvement setting and, hence, it does not consider the existence of *conflicting mis*, and how to determine which of them should be applied; (3) In the context of an iterative process, *conflicting mis* may be added in distinct execution steps, and, hence, the determination of the resulting tuples has to take that into consideration.

4. ADDRESSING THE CONFLICTING PROBLEM AMONG MANUAL DATA REPAIR INSTANCES

As mentioned above, [10] does not address neither the involvement of multiple users in the data cleaning process

bmi	value	based	confidence	expertise
bmi1	office="Loures"	address="Loures" $\wedge$ type="IND"	0.8	0.7
bmi2	office="Lisboa"	address="Loures" $\wedge$ type="IND"	0.6	0.7
bmi3	address="Porto"	office="Porto"	0.5	0.7
bmi4	name="José Vieira Andrade"	office="Loures" $\wedge$ name="José V. A." $\wedge$ address="Loures"	0.4	0.7
bmi5	type="COL"	office="Loures"	0.2	0.4
bmi6	name="José Vasco André"	name="José V. A." $\wedge$ date="5/1974" $\wedge$ office="Loures"	0.6	0.6
bmi7	office="Lisboa"	name="José Vieira Andrade"	0.9	0.9
bmi8	ssn="150111111"	name="José Vasco André" $\wedge$ address="Loures"	0.7	0.7

Figure 6: The *bmis* wrt tuple 7 of table R1, Mdr1, and the DCG in Figure 2 - value **denotes the update of the *bmi*, i.e. the updated attribute and its new value.** based **denotes the *justification* of the *bmi*.** confidence **and** expertise **denote the user-confidence and user-expertise values associated with respective *bmi*.**

nor the possible existence of *conflicting mis*. Moreover, the application order of *mis* during the execution of the DCG is not defined. Therefore, is not clear what should be the result of the execution of our DCG for the Entities table considering a set of possible *conflicting mis*. In order to overcome these problems, we need to: (1) determine which *mis* can be applied together, i.e. non-conflicting *mis*; and (2) determine which sets of non-conflicting *mis* are applied in the corresponding execution step, and in which order.

For this purpose, first, we introduce a new type of *mi*, called *based manual data repair instance*, or *bmi* for short. A *bmi* enables the user to add the *justification* of his/her repair decision using propositions concerning the attribute values of the tuple at hand. This way, a *bmi*, for instance, can extend mi3 by including the *justification* given by user $U3$ to update the address value of tuple 7 to "Porto", i.e. office is "Porto". *Bmis* are also associated with two weight values that represent user's confidence and expertise levels. Based on these values, we determine the most *reliable bmis*, i.e., in the sense that they were added by the most *reliable* users, and, therefore, they should be applied.

Figure 6 illustrates some *bmis* that were introduced by multiple users in different execution steps of the data cleaning process applied to the Entities table wrt tuple 7 of table R1. As said above, one of the purposes of these *bmis* is to update tuple 7 in such way that the resulting tuple satisfies the quality constraint Qc1. However, it also possible to update attribute values that satisfy the quality constraints if the user believes that those values are incorrect. In our running example, we assume that bmi1, bmi2 and bmi3 were added in the first execution step; bmi4, bmi5 and bmi6 were added in the second execution step; bmi7 was added in the third execution step; and, finally, bmi8 was added in the fourth execution step.

Notice that bmi1, bmi2 and bmi3 extend the *mis* mi1, mi2 and mi3, respectively. For instance, bmi1 enables the user $U1$ to represent a manual update of attribute office of tuple 7 and also the *justification* of his/her decision. In this context, address="Loures" and type="IND" denote that the reason why the updated action was office="Loures" is because the address and type values of the respective tuple is "Loures" and "IND", respectively.

Second, we introduce the notion of *conflicting bmis*. This notion allows us to determine which *bmis* can be applied together. In Section 3, we noticed that *mis* can be *conflicting* if they have opposite or incompatible repair actions to the same tuple or if they have incompatible repairs actions and *justifications*. Since the notion of *bmi* includes a *justification* for the manual repair, we are able to capture both causes of conflict. For instance, bmi3 is in *conflict* with bmi4 because bmi3 updates the address value to "Porto" and bmi4 justifies its action by assuming that the address value is "Loures".

Third, we introduce an *argumentation framework* to automatically determine the sets of based manual data repair instances that should be applied in each execution step of the DCG. Based on the notion of *conflict* between *bmis* and the notion of *weight* of a *bmi*, we use argument acceptability semantics to determine the most *reliable* sets of *bmis* to be applied during the execution of the data cleaning process.

Figure 7 illustrates the execution steps of the data cleaning process wrt tuple 7 of table R1 and the corresponding *bmis* that were added and applied during each execution step of our running example, which were automatically determined by our argumentation framework. It also shows, for each execution step, the tuple that resulted from applying the selected *bmis* to tuple 7 of table R1. Notice that, in our example, each execution step performs a total execution of the data transformations. In the following, we briefly described the contents of Figure 7.

In execution step 1, bmi1, bmi2 and bmi3 were added by users. Since these *bmis* are all in *conflict* with each other, they cannot be applied together and, therefore, only one of them can be applied. In this case, bmi1 was selected to be applied to tuple 7 because it was determined as the most *reliable bmi* based on its confidence and expertise levels, i.e. it has the highest values of these levels. Hence, the resulting tuple denoted by t_1 results from the application of bmi1 to t_0, the initial tuple 7 of table R1.

In execution step 2, R1 includes the tuple t_1 instead of t_0, which was updated by the application of bmi1. Since t_1 violates Qc1, it continues to be a blamed tuple and, hence, included in the view associated with Mdr1. For this reason, bmis targeting tuple 7 can be added by users. In this case, bmi4, bmi5 and bmi6 were added by users to be applied to t_1. Moreover, bmi1 and bmi4 were selected to be applied because there were determined as the most *reliable bmis* that can be applied together, i.e. *non-conflicting*. The resulting tuple denoted by t_2 results from the their application to t_0.

In execution step 3, R1 includes the tuple t_2 instead of t_1. Since t_2 violates Qc1, it is included in the view associated with Mdr1. In this case, bmi7 was added by a user to be applied to t_2. This user believes the office of the entity represented by tuple 7 should be "Lisboa" because its name is "José Vieira Andrade". However, bmi7 is in conflict with bmi1 and bmi4. Since t_2 resulted from the application of bmi1 and bmi4 to t_0 and given the fact that bmi7 is more *reliable* than bmi1 and bmi4, the set of applied *bmis* was redetermined without both of these *bmis*. In this case, bmi7 is "used" to prove that bmi1 and bmi4, together, correspond to incorrect repair actions. As a result, the selected bmis to be applied over t_0 were bmi1 and bmi6 and, therefore, the resulting tuple of this execution step was t_3.

In the last execution step 4, R1 includes the tuple t_3 instead of t_2. Since t_3 violates Qc1, because it has a null ssn

Execution step	Added *bmis*	Applied *bmis*	Resulting tuple
0	$\emptyset$	$\emptyset$	$t_0 =\langle$7, "José V. A.", null, 5/1974, IND, "Loures", "Porto"$\rangle$
1	{bmi1, bmi2, bmi3}	{bmi1}	$t_1 = \langle$7, "José V. A.", null, 5/1974, IND, "Loures", "Loures"$\rangle$
2	{bmi4, bmi5, bmi6}	{bmi1,bmi4}	$t_2 = \langle$7, "José Vieira Andrade", null, 5/1974, IND, "Loures", "Loures"$\rangle$
3	{bmi7}	{bmi1,bmi6}	$t_3 = \langle$7, "José Vasco André", null, 5/1974, IND, "Loures", "Loures"$\rangle$
4	{bmi8}	{bmi1,bmi6,bmi8}	$t_4 = \langle$7, "José Vasco André", 150111111, 5/1974, IND, "Loures", "Loures"$\rangle$

Figure 7: Execution steps wrt tuple 7 of table R1 and the DCG of Figure 2, where execution step 0 correspond to the initial tuple 7. Added *bmis* **corresponds to the set of *bmis* added by users during the execution step.** Applied *bmis* **corresponds to the set of *bmis* applied during the execution step.** Resulting tuple **corresponds to the tuple that resulted from applying the set of *bmis* in** Applied *bmis* **column to the initial tuple 7.**

value, t_3 is included in the view associated with Mdr1. In this execution step, bmi8 was added by a user to be applied to t_3. Since bmi8 is not in *conflict* with bmi1 and bmi6, bmi1, bmi6 and bmi8 were applied to t_0 that resulted in tuple t_4. This resulting tuple satisfies the constraint Qc1 and, therefore, does not appear in the the view associated with Mdr1 in the following execution steps of the data cleaning process.

In the following sections, we introduce the notions that support the computation of the set of Applied *bmis* in each execution step of the data cleaning process.

5. BASED MANUAL DATA REPAIR INSTANCES

Terminology. A domain D is a set of atomic values. We assume a set $\mathcal{D}$ of domains and a set $\mathcal{A}$ of names – attribute names – together with a function Dom: $\mathcal{A} \rightarrow \mathcal{D}$ that associates domains to attributes. We consider a set $\mathcal{R}$ of relations names and, for every $R \in \mathcal{R}$ a relation schema $sch(R)$ constituted by an ordered set $A_1, ..., A_n$ of attribute names. We write $R(A_1, ..., A_n)$. Given a relation schema $R = A_1, ..., A_n$, a R-tuple t is an element of $Dom(A_1) \times ... \times Dom(A_n)$. An *instance* of a relation R is a finite set of $sch(R)$-tuples. In the following, we assume that every relation R has a key that can not be updated by a *bmi*. Moreover, the key value of a R-tuple t is given by $key(t)$.

During the iterative data cleaning process, the accuracy of the attribute values of a tuple is expected to increase. In some attribute domains, the developers of the DCG are able to define an order between domain values that follows that accuracy increase. We name that order as a *quality preference* and define it as follows:

Definition 5.1. *Let $R(A_1, ..., A_n)$ be a relation schema.*

- *A **quality preference** wrt A_i is a partial order $\rhd_{A_i} \subseteq Dom(A_i) \times Dom(A_i)$ such that $v \rhd_{A_i}$ null for every $v \in Dom(A_i)$, where $1 \leq i \leq n$.*
- *Given $v_1, v_2 \in Dom(A_i)$:*
  - *v_1 is **preferable** to v_2 wrt $\rhd_{A_i}$ if $v_1 \rhd_{A_i} v_2$.*
  - *We write $v_1 \diamond_{A_i} v_2$ if $v_1 \rhd_{A_i} v_2$ or $v_2 \rhd_{A_i} v_1$. If $v_1 \diamond_{A_i} v_2$ we say that v_1 and v_2 are **related** values. Otherwise, we say that they are **non-related** values.*

A **quality preference** wrt an attribute is a partial order among its domain, where every value of its domain is preferable than *null*. Moreover, two attribute values are non-related if there is no quality preference relation between them.

Example 5.1. *Let $\rhd_{eid}$, $\rhd_{name}$, $\rhd_{ssn}$, $\rhd_{date}$, $\rhd_{type}$, $\rhd_{address}$ and $\rhd_{office}$ be quality preferences over* eid, name, ssn, date, type, address *and* office *attributes, respectively, such that:*

- *$v_1 \rhd' v_2$, s. t. $\rhd' \in \{\rhd_{eid}, \rhd_{ssn}, \rhd_{type}, \rhd_{office}\}$, if $v_1 = v_2$.*
- *$v_1 \rhd_{name} v_2$, $v_1 \rhd_{date} v_2$ and $v_1 \rhd_{address} v_2$ if v_1 has a more complete name, date and address than v_2, respectively.*

We have that:

- *"José Vieira Andrade" $\rhd_{name}$ "José V. Andrade" because "José Vieira Andrade" has a more complete middle name than "José V. Andrade". Hence, "José V. Andrade" $\not\rhd_{name}$ "José Vieira Andrade" and "José Vieira Andrade" $\diamond_{name}$ "José V. Andrade".*
- *"José V. André" $\not\rhd_{name}$ "José V. Andrade" and "José V. Andrade" $\not\rhd_{name}$ "José V. André". Hence, "José V. André" $\not\diamond_{name}$ "José V. Andrade".*
- *"150691111" $\not\rhd_{ssn}$ "500726666" and "150691111" $\rhd_{ssn}$ "150691111" because there is no preference between different ssns. Hence, "150691111" $\not\diamond_{ssn}$ "500726666" and "150691111" $\diamond_{ssn}$ "150691111".*

In the remaining of this paper, we assume that every considered attribute domain is associated with a quality preference order. In the following, we recall the definitions of *manual data repairs* and *manual data repair instances* introduced in [10].

Definition 5.2. *[10] A **Manual Data Repair** m over a relation $R(A_1, ..., A_n)$ consists of a pair $\langle view(m), action(m)\rangle$, where $view(m)$ is an updatable view over R and $action(m)$ is one of the actions that can be performed over $view(m)$, where $1 \leq i \leq n$: action ::=* **delete** | **insert** | **update** A_i

We use relation(m) *to refer to R, and, in the case where the action is* **update** A_i, attribute(m) *to refer to A_i.*

Figure 3 illustrates the *mdr* associated with R1. For an easier reading, Mdr1 uses a natural extension of Definition 5.2 that allows multiple actions and attribute updates.

Definition 5.3. *[10] Let m be a mdr. If $action(m)$ is* **delete** *or* **insert***, an m-instance ι is a pair $\langle m, tuple(\iota)\rangle$ where $tuple(\iota)$ is a $view(m)$-tuple. If $action(m)$ is* **update** *A, an m-instance is a triple $\langle m, tuple(\iota), value(\iota)\rangle$ where $tuple(\iota)$ is a $view(m)$-tuple, and $value(t)$ is a value in $Dom(A)$.*

As mentioned in above, the *mis* do not take into consideration possible *conflicts* between them and the existence of multiple users. In order to address these issues, we define the notion of *based manual data repair instance* that extends the notion of *mi* by adding a *justification* to the correction action, the user's confidence and expertise levels.

Definition 5.4. *Let m be a manual data repair. A **based mdr-instance** of m, or bmi of m, ρ is a tuple $\langle m, tuplek(\rho)$, $value(\rho), based(\rho), confidence(\rho), level(\rho)\rangle$, where:*

(1) $tuplek(\rho)$ is the tuple key.

(2) $value(\rho)$ is:

   (a) *a value in $Dom(A)$ if $action(m)$ is* **update** *A; or*
   (b) *$\emptyset$ if $action(m)$ is* **delete***; or*
   (c) *a $relation(m)$-tuple with key equal to $tuplek(\rho)$ if $action(m)$ is* **insert***.*

(3) *$based(\rho)$ is:*
   (a) *a first-order formula, with one free variable denoted by the tuple $\bar{t}$, defined over the attributes of $view(m)$ if $action(m)$ is* **update** *or* **delete***; or*
   (b) *$\emptyset$, otherwise.*

(4) *$confidence(\rho) \in [0,1]$ denotes the user confidence level.*

(5) *$expertise(\rho) \in [0,1]$ denotes the user expertise level.*

Given a *bmi* ρ, $tuplek(\rho)$ is the key of the tuple, named *target tuple*, with schema of $relation(\rho)$, to which ρ's action will be applied. In our setting, we denote the *target tuple* as $\bar{t}$. Notice that, in opposition to a *mi*, a *bmi* is associated with a key and not with a tuple. This way, we are able to apply a *bmi* in different execution steps where the tuple with key $tuplek(\rho)$ may have distinct attribute values.

The value of ρ, $value(\rho)$, represents the changes that ρ suggests to be applied. If $action(\rho)$ is **update** A then $value(\rho)$ is the updated value for attribute A. If $action(\rho)$ is **delete** then $value(\rho)$ is $\emptyset$. If $action(\rho)$ is **insert** then $value(\rho)$ is the tuple with key $tuplek(\rho)$ that will be inserted in $relation(\rho)$.

$based(\rho)$ is a first-order formula that denotes the *justification* for ρ, i.e., which propositions justify the user intervention. Notice that $based(\rho)$ is only related to the attribute values of the target tuple, i.e., the ρ action is only decided based on the attribute values of the target tuple. Moreover, if $action(\rho)$ is **insert** then $based(\rho)$ is an empty set because ρ cannot be justified based on an inexisting tuple. $confidence(\rho)$ denotes the user-confidence that is related to ρ and is defined by the corresponding user. Finally, $expertise(\rho)$ denotes the user expertise level. This value is usually set by the developers of the corresponding DCG.

Example 5.2. *Figure 6 illustrates some examples of bmis. For instance,* bmi1 *denotes a* bmi *of $Mdr1$ related with tuple 7 where the user proposes to update the value of* office *to "Loures" because the* address *value is "Loures" and* type *value is "IND". Moreover, the user-confidence and the user-expertise is 0.8 and 0.7, respectively.*

Given a *bmi* ρ, we define the **weight** of ρ as a function $W : \rho \rightarrow [0,1]$ that depends and is monotonic wrt $confidence(\rho)$ and $expertise(\rho)$ values. The weight of a *bmi* denotes a quality value based on its confidence and expertise values. This value is used to compute the *reliability* of a *bmi*.

Example 5.3. *Let $W1$ and $W2$ be weight functions such that $W1(\rho) = (expertise(\rho) * (1 + confidence(\rho)))/2$ and $W2(\rho) = (confidence(\rho) + 2*(expertise(\rho)))/3$. $W1(\mathsf{bmi1}) = 0.63$, $W1(\mathsf{bmi2}) = 0.56$ and $W1(\mathsf{bmi6}) = 0.48$, $W2(\mathsf{bmi1}) = 0.73$, $W2(\mathsf{bmi2}) = 0.67$ and $W2(\mathsf{bmi6}) = 0.6$.*

In Section 3, we noticed that *mis* can be *conflicting* and, in those cases, should not be applied together. For this purpose, first, we introduce the notion of *bmi applicable to a tuple* and, second, introduce the notion of *conflict* pair of *bmis*. For the sake of simplicity, in our formalization, we define a *null tuple* as a tuple with a null value in all its attributes, which represents a tuple that is not in a table. In an envisaged implementation of our framework this notion is not materialized.

Definition 5.5. *Let ρ be a bmi and t a tuple with schema of $relation(\rho)$, $R(A_1, ..., A_n)$, s.t. $t[A_1, ..., A_n] = \langle a_1, ..., a_n \rangle$. We say that ρ is* ***applicable to*** *t if:*

(1) *if $action(\rho)$ is* insert *then t is a null tuple; otherwise, $key(t) = tupleK(\rho)$.*

(2) *$based(\rho) \cup \{\bar{t}[A_1] = a_1, ..., \bar{t}[A_n] = a_n\}$ is consistent.*

We denote by $\rho(t)$ the R-tuple resulting from applying ρ over a tuple t. If $action(\rho)$ is delete *then $\rho(t)$ is a null tuple. If $action(\rho)$ is* insert *then $\rho(t) = value(\rho)$. If $action(\rho)$ is* update A*, then $\rho(t)$ is the tuple resulting from updating $t[A]$ to $value(\rho)$. Finally, if ρ is not applicable to t then $\rho(t) = t$.*

A non-insertion *bmi* ρ is applicable to a given tuple t if: (1) t is the target tuple of ρ; and (2) the attribute values of t are consistent with $based(\rho)$. If ρ denotes a tuple insertion, then ρ is applicable to t if t is a null tuple.

Example 5.4.

- bmi1 *is applicable to $t_0 = \langle$7, "José V. A.", null, 5/1974, IND, "Loures", "Porto"$\rangle$.*
- bmi1 *is not applicable to tuple 8 of the* R1 *table, t', because $tupleK(\mathsf{bmi1}) = 7$ and $key(t') = 8$. Therefore, it fails condition (1) of Definition 5.5.*

Given a set of *bmis* $P = \{\rho_1, ..., \rho_n\}$, an application order of P, $P^a = \langle \rho'_1, ..., \rho'_n \rangle$, is a sequence of *bmis* such that $\rho'_i \in P$ and $1 \leq i \leq n$. Moreover, we denote by $P^a(t)$ the resulting tuple $\rho'_n(\rho'_{n-1}(...(\rho'_1(t))...))$.

Given two application orders over the same tuple t and their corresponding resulting tuples, t_1 and t_2, it is useful to determine if t_1 is preferable to t_2 (in the sense of Definition 5.1). To do that, we need to take in consideration only attribute values of t_1 and t_2 that are distinct from t, i.e. the attributes values that were updated by the corresponding sets of *bmis*. For this purpose, we introduce the following definition:

Definition 5.6. *Let t, t_1 and t_2 be tuples with schema $R(A_1, ..., A_n)$. We write:*

- *$t_1 \triangleright_t t_2$ if $t_1[A_i] \triangleright_{A_i} t_2[A_i]$ or $t_2[A_i] = t[A_i]$, for $1 \leq i \leq n$.*

  $t_1 \triangleright_t t_2$ denotes that t_1 is ***preferable*** *to t_2 wrt t, i.e. every (updated) attribute value of t_1 is preferable to the corresponding attribute value of t_2 wrt to the initial tuple t.*

- *$t_1 \diamond_t t_2$ if $t_1 \triangleright_t t_2$ or $t_2 \triangleright_t t_1$.*

  $t_1 \diamond_t t_2$ denotes that t_1 is ***non-contradictory*** *to t_2 wrt t, i.e. every (updated) attribute value of t_1 is related, wrt a quality preference order, to the corresponding attribute value of t_2 wrt to the initial tuple t.*

Example 5.5. *Let t_0, t_1, t_2, t_3 and t_4 be tuples of Figure 7. We have that: $t_1 \triangleright_{t_0} t_0$ and $t_1 \diamond_{t_0} t_0$; $t_2 \triangleright_{t_0} t_1$, $t_1 \not\triangleright_{t_0} t_2$ and $t_2 \diamond_{t_0} t_1$; $t_3 \not\diamond_{t_0} t_2$, $t_2 \not\diamond_{t_0} t_3$ and $t_3 \not\diamond_{t_0} t_2$; $t_4 \triangleright_{t_0} t_3$, $t_3 \not\triangleright_{t_0} t_4$ and $t_4 \diamond_{t_0} t_3$.*

Definition 5.7. *Let ρ_1 and ρ_2 be distinct bmis. We say that $\langle \rho_1, \rho_2 \rangle$ is a* ***conflicting pair*** *if :*

(1) *$tupleK(\rho_1) = tupleK(\rho_2)$; and*

(2) (a) *$action(\rho_1) \neq action(\rho_2)$ and, $action(\rho_1) =$* **delete** *or $action(\rho_2) =$* **delete***; or,*

(b) *$action(\rho_1) = action(\rho_2) =$* **update** *$A$ and $value(\rho_1) \not\succ_A value(\rho_2)$; or,*

(c) *$action(\rho_1) =$* **update** *A, $action(\rho_2) =$* **update** *B, $A \neq B$, and $based(\rho_2) \cup \{\bar{t}[A] \diamond_A value(\rho_1)\}$ is inconsistent.*

We say that a set of *bmis* P is a **conflicting set** if there is a conflicting pair $\langle \rho_i, \rho_j \rangle$ s.t. $\rho_i, \rho_j \in P$. Otherwise, P is a **conflict-free** set.

Two *bmis*, ρ_1 and ρ_2, form a conflicting pair $\langle \rho_1, \rho_2 \rangle$ if: (1) they are associated with the same tuple's key; and, ($2a$) define distinct actions to be applied and one of them is a delete action. The deletion of a tuple is always in conflict with an insertion or modification of the same tuple. In our setting, we assume that an insertion cannot be in conflict with an update action, because an inserted tuple can also be updated; or, ($2b$) they update the same attribute to different values that do not have a quality preference relationship between them, i.e. they are non-related values; or, ($2c$) they apply updates to different attributes, and the result of applying ρ_1 is in conflict with the *justification* of ρ_2 and, therefore, in conflict with its application. Notice that condition ($2c$) ensures that a conflicting pair is asymmetric. Moreover, a conflicting pair $\langle \rho_1, \rho_2 \rangle$ can be determined independently of the contents of the target tuples because Definition 5.7 only depends on their key values (Condition (1)).

Example 5.6.

- $\langle$bmi1, bmi2$\rangle$ *and* $\langle$bmi2, bmi1$\rangle$ *are conflicting pairs because they satisfy conditions* (1) *and* ($2b$) *of Definition 5.7, because $value($bmi1$) \not\succ_{office} value($bmi2$)$, i.e. "Loures" $\not\succ_{\text{office}}$ "Lisboa".*
- $\langle$bmi2, bmi3$\rangle$ *and* $\langle$bmi3, bmi2$\rangle$ *are conflicting pairs because they satisfy conditions* (1) *and* ($2c$) *of Definition 5.7.*

Figure 8 illustrates a graph that represents all the conflicting pairs between the bmis *illustrated in Figure 6. Each arrow of the graph denotes a conflicting pair between the two connected nodes.*

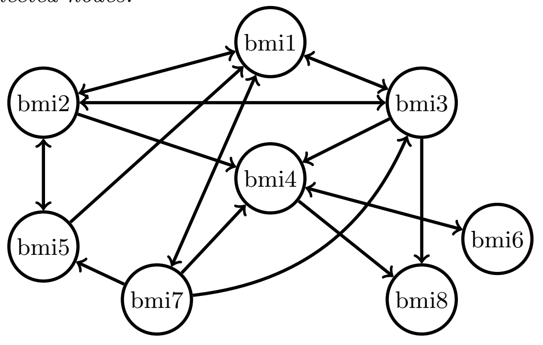


Figure 8: *Bmis* conflict graph.

As said above, an iterative data cleaning process may have several execution steps. Thus, distinct *bmis* may be introduced and applied in different execution steps. For this purpose, we introduce the notion of applicable set of *bmis* to determine which *bmis* can be applied together.

Definition 5.8. *Let $P = \{\rho_1, ..., \rho_n\}$ be a set of non-empty* bmis *and t a tuple. We say that P is* ***applicable*** *to t if :*

(1) P is a conflict-free set; and,

(2) there is an application order $P^a = \langle \rho'_1, ..., \rho'_n \rangle$ of P such that, for $1 \leq i \leq n-1$, we have that: (a) $t'' \rhd_t t'$ and (b) $t'' \neq t'$, where $t'' = \rho'_{i+1}(...(\rho'_1(t))...)$ and $t' = \rho'_i(...(\rho'_1(t))...)$.

We denote by $P(t)$ the resulting tuple $P^a(t)$ and $tupleK(P)$ as $tupleK(\rho_1)$.

Given a set of *bmis* P and a tuple t, P is applicable to t if: does not have conflicting pairs (Condition (1)); and, there is a application order, P^a, of P where every application of a *bmi* produces a different (Condition ($2b$)) and more preferable (Condition ($2a$)) tuple wrt t; Condition ($2b$) guarantees that P is minimal with respect to the resulting tuple $P^a(t)$, i.e. every *bmi* of P is needed to produce the resulting tuple $P(t)$. Given that P^a always exists and it is unique [14], the application order of the *bmis* in an applicable set is pre-determined.

Example 5.7. *Let $t_0 = \langle$7, "José V. A.", null, 5/1974, IND, "Loures", "Porto"$\rangle$ be the tuple 7 of* R1 *table as illustrated in Figure 7.*

- *$P_1 = \{$bmi1, bmi4$\}$ is applicable to tuple t_0, because it verifies the conditions of Definition 5.8. Moreover, $P_1(t_0) = \langle$7, "José Vieira Andrade", null, 5/1974, IND, Loures, Loures$\rangle$.*
- *$P_2 = \{$bmi1, bmi2$\}$ is not applicable to t_0 because $\langle$bmi2, bmi1$\rangle$ is a conflicting pair and, therefore, it fails on Condition* (1) *of Definition 5.8.*
- *$P_3 = \{$bmi4, bmi7$\}$ is not applicable to t_0 because neither* bmi4 *nor* bmi7 *is applicable to t_0 and, therefore, for every application order P^a for P we have that $P^a(t_0) = t_0$. Hence, it fails on Condition* ($2b$) *of Definition 5.8.*

Distinct applicable sets cannot be applied simultaneously if they have *conflicting bmis*. Given the previous definitions and notation, the following proposition shows a connection between two *conflicting* applicable sets.

Proposition 5.1. *Let t and t' be tuples with schema $R(A_1, ..., A_n)$, P and P' be applicable sets to t and t', respectively, where $tupleK(P) = tupleK(P')$, and $P'(t')[A_1, ..., A_n] = \langle a_1, ..., a_n \rangle$.* (1) *iff* (2) *or* (3), *where:*

(1) There are $\rho \in P$ and $\rho' \in P'$ such that $\langle \rho', \rho \rangle$ is a conflicting pair.

(2) $P'(t') \not\succ_t P(t)$.

(3) There is $\rho'' \in P$ such that :

(a) If $action(\rho'') =$ **update** *A_i then $based(\rho'') \cup \{\bar{t}[A_1] = a_1, ..., \bar{t}[A_{i-1}] = a_{i-1}, \bar{t}[A_{i+1}] = a_{i+1}, ..., \bar{t}[A_n] = a_n\}$ is inconsistent.*

(b) Otherwise, $based(\rho'') \cup \{\bar{t}[A_1] = a_1, ..., \bar{t}[A_n] = a_n\}$ is inconsistent.

The previous proposition shows that two applicable sets wrt the same tuple's key are in *conflict*, i.e. their union is a conflict-set (Condition (1)), if only if their resulting tuples are contradictory (Condition (2)); or the resulting tuple of one of them is contradictory with the justifications of the other (Condition (3)).

6. ARGUMENTATION FRAMEWORK

Based on the notion of conflicting pair, we introduced the notion of applicable set of *bmis* to determine which *bmis* can be applied together to a given tuple. In order to determine which applicable set should be applied, we need to find some criteria that allows us to distinguish and compare distinct applicable sets. In this section, we present such a criteria based on the argument acceptability criteria presented in [7, 5]. We construct an argumentation framework (AF , for short) based on applicable sets by considering a *bmi* as an application rule with respect to its action and an applicable set as an argument. Moreover, an applicable set P to a given tuple t is viewed as an argument in favor of the resulting tuple $P(t)$, where each *bmi* in P compose the support of that argument. Since a *bmi* is associated with a weight, it allows us to evaluate and compare distinct arguments. In this section, we use $A, B, C, ...$ to denote arguments.

DEFINITION 6.1. *Let P be a set of bmis and t a tuple. An* ***argument*** *is a pair $\langle P, t\rangle$ such that P is applicable set to t. We say that $\langle P, t\rangle$ is an argument for $P(t)$, that $P(t)$ is the conclusion of the argument and P and t are the support of the argument, respectively. Moreover, if $A = \langle P, t\rangle$ is an argument, we define: $SupB(A) = P$, $SupT(A) = t$, $Conc(A) = P(t)$ and $tupleK(A) = tupleK(P)$.*

As in [1] an argument $\langle P, t\rangle$ has a minimum and consistent support because the support is an applicable set with respect to $P(t)$. Since the conclusion of an argument is the unique resulting tuple of the support, the conclusion is omitted from the pair that constitutes an argument.

EXAMPLE 6.1. *Lets consider the bmis illustrated in Figure 6 and tuple 7, t_0, of* R1 *illustrated in Figure 7, we have that:*

- $A = \langle\{\mathsf{bmi1}\}, t_0\rangle$, $B = \langle\{\mathsf{bmi2}\}, t_0\rangle$, $C = \langle\{\mathsf{bmi3}\}, t_0\rangle$, $D = \langle\{\mathsf{bmi1}, \mathsf{bmi4}\}, t_0\rangle$, $E = \langle\{\mathsf{bmi1}, \mathsf{bmi6}\}, t_0\rangle$ *and* $G = \langle\{\mathsf{bmi1}, \mathsf{bmi6}, \mathsf{bmi8}\}, t_0\rangle$ *are arguments. The support of these arguments represent all the existing applicable sets to t_0.*
- $\langle\{\mathsf{bmi1}, \mathsf{bmi2}\}, t_0\rangle$ *is not an argument because* $\{\mathsf{bmi1}, \mathsf{bmi2}\}$ *is not an applicable set to t_0.*
- *There is no argument $\langle P, t_0\rangle$ where* $\mathsf{bmi5} \in P$ *because there is no applicable set to t_0 that includes* bmi5*.*
- $F = \langle\{\mathsf{bmi7}\}, Conc(D)\rangle$ *and* $H = \langle\{\mathsf{bmi5}\}, Conc(A)\rangle$ *are also arguments.*

Given the notion of argument, we define the notion of counter-argument as follows:

DEFINITION 6.2. *Let $A = \langle P, t\rangle$ and $B = \langle P', t'\rangle$ be arguments. B is a* ***counter-argument*** *of A if (1) there are $\rho \in P$ and $\rho' \in P'$ such that $\langle \rho', \rho\rangle$ is a conflicting pair; and, (2a) $t = t'$; or, (2b) there is $\emptyset \subset Q \subseteq P$ such that $t' = Q(t)$.*

In our context, argument A and a counter-argument B of A represent applicable sets that are in *conflict*. Given the result of Proposition 5.1, our notion of counter-argument includes: (1) the notion of *rebuttal* in the sense of [1], i.e. an argument with a contradictory conclusion of the targeted argument. This is shown based on Condition (2a) of Definition 6.2 and Condition (2a) of Proposition 5.1; and, (2) the notion of *undercut* in the sense of [1], i.e. an argument that contradicts the support of targeted argument. This is shown based on Condition (2a) of Definition 6.2 and Condition (2b) of Proposition 5.1; and, finally, (3) the notion of *HY-argument* in the sense of [4], i.e. an argument that shows that the targeted argument leads to a contradiction. This is shown based on Condition (2b) of Definition 6.2 and Condition (2a) of Proposition 5.1;

EXAMPLE 6.2. *Let us consider the bmis illustrated in Figure 6 and tuple t_0 illustrated in Figure 7.*

- $A = \langle\{bmi1\}, t_0\rangle$ *is a counter-argument of* $B = \langle\{bmi2\}, t_0\rangle$ *because they satisfy Condition (1a) of Definition 6.2. A is also a* rebuttal *of B in the sense of [1].*
- $C = \langle\{bmi3\}, t_0\rangle$ *is counter-argument of* $A = \langle\{bmi2\}, t_0\rangle$*. C is also a* undercut *of A in the sense of [1].*
- $F = \langle\{bmi7\}, Conc(D)\rangle$ *is a counter-argument of* $D = \langle\{bmi1, bmi4\}, t_0\rangle$ *because they satisfy Condition (1b) of Definition 6.2. F is also a* HY-argument *of D in the sense of [4].*

We consider an abstract argumentation framework $\langle \mathcal{A}r, \mathbf{def}\rangle$ where $\mathcal{A}r$ is a set of arguments as defined in Definition 6.1 and $\mathbf{def} \subseteq \mathcal{A}r \times \mathcal{A}r$ denotes a defeat relation between arguments. In our setting, a defeat relation must ensure that (1) if A_1 **def** A_2 then A_1 is a counter-argument of A_2 and (2) if A_1 is a counter-argument of A_2 and A_2 is a counter-argument of A_1 then A_1 **def** A_2 or A_2 **def** A_1.

In the context of an execution step and the target tuple of a set of *bmis*, t, an AF is constructed by: (1) defining a defeat relation; (2) including the arguments that are supported in tuple t and in a set of *bmis* already added; (3) including all the arguments that defeat an argument of the later set of arguments.

EXAMPLE 6.3. *Let* $\boldsymbol{def}_1$ *and* $\boldsymbol{def}_2$ *be defeat relations s. t.: A_1 $\boldsymbol{def}_1$ A_2 and A_1 $\boldsymbol{def}_2$ A_2 if A_1 is a counter-argument of A_2 and $Min(\{W(\rho) \mid \rho \in SupB(A_1)\}) \geq Min(\{W(\rho) \mid \rho \in SupB(A_2)\})$, where W is $W1$ and $W2$, respectively. Moreover, let Af_1, Af_2, Af_3, Af_4, Af_5 be* AFs *such that:*

- $Af_1 = \langle \mathcal{A}_1, \boldsymbol{def}_1\rangle$*, where* $\mathcal{A}_1 = \{A, B, C\}$.
- $Af_2 = \langle \mathcal{A}_2, \boldsymbol{def}_1\rangle$*, where* $\mathcal{A}_2 = \{A, B, C, D, E\}$.
- $Af_3 = \langle \mathcal{A}_3, \boldsymbol{def}_1\rangle$*, where* $\mathcal{A}_3 = \{A, B, C, D, E, F\}$.
- $Af_4 = \langle \mathcal{A}_4, \boldsymbol{def}_1\rangle$*, where* $\mathcal{A}_4 = \{A, B, C, D, E, F, G\}$.
- $Af_5 = \langle \mathcal{A}_2, \boldsymbol{def}_2\rangle$.

Figure 9 illustrates the defeat argumentation graph wrt $\boldsymbol{def}_1$ *and* $\boldsymbol{def}_2$*. The defeat argumentation graph wrt each of previous* AFs *corresponds to the subgraph of Figure 9 that includes the respective set of arguments and defeat relation.*

Af_1, Af_2, Af_3 and Af_4 correspond to the AFs *that were constructed in execution steps 1, 2, 3 and 4 illustrated in Figure 7, respectively. Af_5 was constructed as example only.*

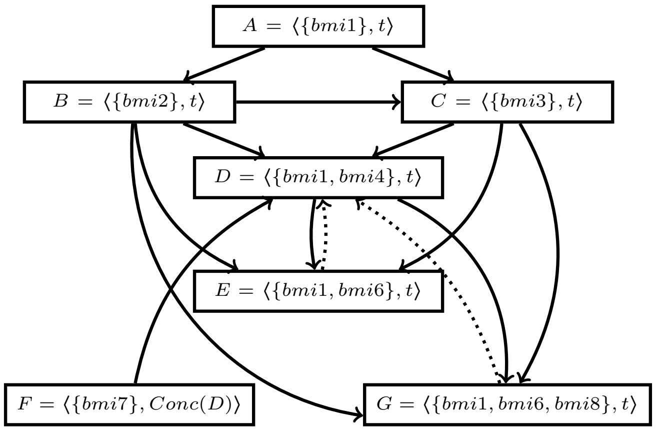
Af_1 includes all the arguments that can be constructed based on the bmis bmi1, bmi2 *and* bmi3 *that were added in execution step 1 wrt tuple 7 of table* R1*.*

Af_2 includes all the arguments that can be constructed based on the bmis *added until execution step 2 wrt tuple 7 of table* R1*. That is,* bmi1*,* bmi2*,* bmi3*,* bmi4*,* bmi5 *and* bmi6*. Notice that, since* bmi5 *is not a member of any applicable set to tuple 7, there is no argument with* bmi5 *in its support.*

Af_3 includes all the arguments that can be constructed based on the bmis *added in execution step 3, i.e.* bmi7*, and*

in previous execution steps. With bmi7 *we are able to construct the argument, F, that defeats the argument D. Therefore, F is also included in Af_3.*

Af_4 includes all the arguments that were constructed in the previous execution steps and the argument G that includes bmi8*, which was added in execution step 4.*



$A = \langle\{bmi1\}, t\rangle$
$B = \langle\{bmi2\}, t\rangle$
$C = \langle\{bmi3\}, t\rangle$
$D = \langle\{bmi1, bmi4\}, t\rangle$
$E = \langle\{bmi1, bmi6\}, t\rangle$
$F = \langle\{bmi7\}, Conc(D)\rangle$
$G = \langle\{bmi1, bmi6, bmi8\}, t\rangle$

Figure 9: The defeat argumentation graph wrt def$_1$, def$_2$ and tuple 7 of R1 - A solid arrow denotes a def$_1$ and a def$_2$ relationship. A dotted arrow denotes a def$_2$ relationship.

In the context of an argumentation framework, the notion of *acceptability* [7, 5] among arguments was introduced to determine which sets of arguments should be "accepted". In our setting, we use this notion to determine which based manual instances should be applied in each execution step of the data cleaning process. In order to introduce some *acceptability* criteria, we recall the notion of *defends*.

Definition 6.3. *[5] Let $\mathcal{A}$ be a set of arguments, $\langle \mathcal{A}, \textbf{def}\rangle$ be an argumentation framework, $A \in \mathcal{A}$ and $Args \subseteq \mathcal{A}$. We define:*

- *$Args^+ = \{B \mid A$ **def** B for some $A \in Args\}$.*
- *$Args$ is a **conflict-free set** if $Args \cap Args^+ = \emptyset$.*
- *$Defs(Args) = \{A \mid \{B \mid B$ **def** $A\} \subseteq Args^+\}$.*

  *We say that $Args$ **defends** A if $A \in Defs(Args)$.*

$Defs(Args)$ stands for the set of arguments that are acceptable with respect to Args in the sense of [7]. Based on the argumentation acceptability semantics introduced in [7, 5], we determine the sets of arguments that should be considered accepted. The sets of *bmis* that compose the support of these arguments represent the *bmis* that should be taken in consideration to be applied in each execution step of the data cleaning process.

Definition 6.4. *Let $\mathcal{A}$ be a set of* bmis *, $\langle \mathcal{A}, \textbf{def}\rangle$ be an argumentation framework, $A \in \mathcal{A}$, $Args \subseteq \mathcal{A}$ a conflict-free set, W a weight function and t a tuple.*

- *[5] $Args$ is a **preferred set** (resp. **grounded set**) if $Args$ is a maximal (resp. minimal) set such that $Defs(Args) = Args$.*
- *t' is the **preferred** (resp. **grounded**) **conclusion** wrt to t if there is $A \in Args$ s.t. $SupT(A) = t$ and $Conc(A) = t'$ and there is no $B \in Args$ s.t. $SupT(B) = t$, $Conc(A) \neq Conc(B)$ and $Conc(B) \rhd_t Conc(A)$, where $Args$ is a preferred (resp. grounded) set.*

The preferred and grounded acceptability semantics can be chosen by the developer to determined which *bmis* are applied in all execution steps. A preferred set of arguments is a maximal set that defends itself and, therefore, guarantees that there are no "better" arguments than those in its set. Notice, however, that it could exist multiple preferred sets, which represent opposite and equally valid alternatives. The developer can introduce extra conditions to determine which preferred set will be selected (e.g., by selecting the preferred sets with the argument with the highest weight value). Instead of selecting one of the preferred sets, the developer can also select the grounded set. The grounded set of arguments is the unique set that includes all the arguments that are in all preferable sets. It also is the minimal set that defends itself. The grounded set represents the set of arguments that are "consensus" among all preferred alternatives.

After choosing which set represents the most *reliable* set of arguments and, therefore, applicable sets, we need to determined the applicable set that will be applied over a given tuple. In preferred and grounded sets, there can be several arguments with different supports, i.e. applicable sets of *bmis*, and different conclusions. We propose to select the most preferable conclusions, i.e. the preferred and grounded conclusions (in [14] we show that these conclusions always exist), respectively, as the resulting tuples.

Example 6.4. *In this example, we use our* AF *to automatically determine the sets of* bmis *that should be applied in the first execution step as it was described in Section 4. (see [14] for a complete example).*

Let t_0 = ⟨7, "José V. A.", null, 5/1974, IND, "Loures", "Porto"⟩ as illustrated in Figure 7. With respect to Af_1 we have that: $Defs(\{A\}) = \{A\}$, $Defs(\{B\}) = \{A\}$ and $Defs(\{C\}) = \{A\}$. Therefore, the unique existing preferred set is $\{A\}$. Since $\{A\}$ is the unique preferred set it is also the grounded set. Given that $\{A\}$ includes only one argument, A, the preferred and grounded conclusion of Af_1 is $Conc(A) = t_1$ = ⟨7, "José V. A.", null, 5/1974, IND, "Loures", "Loures"⟩, where t_1 is illustrated in Figure 7.

	Af1	**Af2**	**Af3**	**Af4**	**Af5**
preferred sets	{A}	{A, D}	{A, E, F}	{A, E, F, G}	{A, D} {A, E}
preferred conclusions	t_1	t_2	t_3	t_4	t_2, t_3
grounded set	{A}	{A, D}	{A, E, F}	{A, E, F, G}	{A}
grounded conclusion	t_1	t_2	t_3	t_4	t_1

Figure 10: Preferred and grounded sets, preferred and grounded conclusions wrt tuple 7 and $Af1$, $Af2$, $Af3$, $Af4$ and $Af5$, where $t_1 = Conc(A)$, $t_2 = Conc(D)$, $t_3 = Conc(E)$ and $t_4 = Conc(G)$ (see Figure 7).

Figure 10 illustrates all the preferred the preferred and grounded sets and respective conclusions wrt the AFs introduced in Example 6.3, i.e. $Af1$, $Af2$, $Af3$, $Af4$ and $Af5$. $Af1$, $Af2$, $Af3$ and $Af4$ correspond to the AFs that were constructed in execution steps 1, 2, 3 and 4 illustrated in Figure 7. In these cases the preferred and grounded coincide because there is only one preferred set. Moreover, the resulting tuples of each execution step of Figure 7 correspond to a grounded conclusion illustrated in Figure 10. Hence, using our AF we are able to automatically determine which *bmis* should be applied in each execution step of the data cleaning process. $Af5$ represents an example of an AF where is more than one preferred set. In these cases, the developers have to add conditions to determined which conclusion must be selected.

7. RELATED WORK

Our proposal extends the work developed in [10], where the user involvement is represented through the notion of *mi*. However, the existence of multiple users and possible conflicting *mis* are not considered. Moreover, the application of *mis* during the data cleaning process is not defined.

The incorporation of user feedback has shown to be useful in several automatic tasks. For example, the data cleaning prototype AJAX [9] supports error handling through the notion of *exception*. These exceptions are stored in specific tables that can be later visualized by the developer and help him/her track the culprit for the exception and apply the required data correction. Potter's Wheel [12] offers a graphical interface through which the developer can specify and quickly debug data cleaning rules that are applied to samples of data. Chai et al [6] propose a solution to incorporate the end-user feedback into Information Extraction programs, which are composed by a set of declarative rules. The developer writes some of these rules with the purpose of specifying the items of data that users can edit and the user interfaces that can be used.

Although these works offer some support for user feedback, they do not consider the context of multiple-users feedback and the possible existence of *conflicts* and the *reliability* of manual data repairs.

Our notion of *bmi* resembles the notion of Conditional Functional Dependency (CFD) [8]. CFDs were introduced in the context of the Constraint Repair Problem [11]. CFDs are introduced by the developer and assumed to be always valid, correct and consistent. In opposition, a *bmi* can be introduced by any user and, hence, *conflicting bmis* can occur. The reliability of a *bmi* depends on the user confidence and expertise level. Moreover, a *bmi* can also include in its *justification* its updated attribute. This is not possible to represent by a CFD because a CFD is denoted by a propositional rule.

Although database repair is closely related to *Belief Revision*, to the best of our knowledge, there is no work that proposes an argumentation-based approach to support the user involvement in the context of a data cleaning process.

8. CONCLUSIONS

In this work, we address the problem of integrating multiple-user feedback in an automatic and iterative data cleaning process. We propose the notion of *bmi* to represent the user feedback required to manually clean data items. This notion extends the notion of *manual data repairs instance* by including a *justification* and the user confidence and expertise level values. We also introduce the notion of *conflict* between *bmis* to identify which sets of *bmis* can be applied simultaneously. Finally, we introduce an AF and use argumentation acceptability criteria to determine which *bmis*, that were added by the users, should be applied during the data cleaning process.

As ongoing work, we are implementing the framework presented in [10] which supports the work presented in this paper. As future work, we envisage the following three types of developments concerning our framework:

First, we plan to extend our framework by associating to each attribute value of every tuple of the input (dirty) tables a *weight* value that denotes its *data quality* value, which can be determined by data provenance procedures as in [3, 8]. These values support the calculation of which *bmis* are the most *reliable* to be applied by taking into account the *quality* of the attribute values that their justifications are based on.

Second, our notion of *bmi* should be extended to a rule that can be applied to any tuple, independently of its key. This way the user can introduce a more general manual data repair that can be applied over several different tuples.

Third, based on [13] proposal, we plan to evaluate each *bmi* wrt the entire DCG. That is, we should take into consideration not only the *conflicts* that a *bmi* may produce in an intermediary table of the DCG but also the rejected tuples that it may produce in the remaining DCG.

9. REFERENCES

[1] P. Besnard and A. Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128(1-2):203–235, 2001.

[2] P. Besnard and A. Hunter. *Elements of Argumentation*. The MIT Press, 2008.

[3] P. Bohannon. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.

[4] M. Caminada. Dialogues and hy-arguments. In *10th International Workshop on Non-Monotonic Reasoning*, pages 94–99, 2004.

[5] M. Caminada and M. W. A. Caminada. On the issue of reinstatement in argumentation. In *JELIA*, pages 111–123, 2006.

[6] X. Chai, B.-Q. Vuong, A. Doan, and J. F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD*, pages 87–100, 2009.

[7] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.

[8] W. Fan, F. Geerts, and X. Jia. Conditional dependencies: A principled approach to improving data quality. In *BNCOD*, pages 8–20, 2009.

[9] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

[10] H. Galhardas, A. Lopes, and E. Santos. Support for user involvement in data cleaning applications. In *DAWAK*, 2011.

[11] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.

[12] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, 2001.

[13] E. Santos. Enhancing a data cleaning process using repairs. *Proceedings of the Third SIGMOD PhD Workshop on Innovative Database Research*, 2009.

[14] E. Santos and H. Galhardas. Using argumentation to support the user involvement in data cleaning. http://web.ist.utl.pt/esantos/arginv11.pdf, 2011.