Argument & Computation 8 (2017) 35–59
DOI 10.3233/AAC-170017
IOS Press

Defeasible argumentation over relational databases

Cristhian Ariel David Deagustini [a,b,*], Santiago Emanuel Fulladoza Dalibón [a,b], Sebastián Gottifredi [a], Marcelo Alejandro Falappa [a,b], Carlos Iván Chesñevar [a] and Guillermo Ricardo Simari [a]

[a] *AI R&D Lab., Institute for Computer Science and Engineering (ICIC), Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Alem 1253, (B8000CPB), Bahía Blanca Bs. As., Argentina*
E-mails: cadd@cs.uns.edu.ar, sef@cs.uns.edu.ar, sg@cs.uns.edu.ar, mfalappa@cs.uns.edu.ar, cic@cs.uns.edu.ar, grs@cs.uns.edu.ar

[b] *Agents and Intelligent Systems Area, Faculty of Management Sciences, Universidad Nacional de Entre Ríos, Tavella 1424, Concordia E. R., Argentina*

Abstract. Defeasible argumentation has been applied successfully in several real-world domains in which it is necessary to handle incomplete and contradictory information. In recent years, there have been interesting attempts to carry out argumentation processes supported by massive repositories developing argumentative reasoning applications. One of such efforts builds arguments by retrieving information from relational databases using the DBI-DeLP framework; this article presents eDBI-DeLP, which extends the original DBI-DeLP framework by providing two novel aspects which refine the interaction between DeLP programs and relational databases. First, we expand the expressiveness of dbi-delp programs by providing ways of controlling how the information in databases is recovered; this is done by introducing filters that enable an improved fine-grained control on the argumentation processes which become useful in applications, providing the semantics and the implementation of such filters. Second, we introduce an argument comparison criterion which can be adjusted at the level of literals to model particular features such as credibility and topic expertise, among others. These new tools can be particularly useful in environments such as medical diagnosis expert systems, decision support systems, or recommender systems based on argumentation, where datasets are often provided in the form of relational databases.

Keywords: Defeasible argumentation, databases, common knowledge repository

1. Introduction

Argumentation represents a sophisticated mechanism for the formalization of commonsense reasoning, which has found application and proven its importance in different areas of Artificial Intelligence (AI) such as legal systems, multi-agent systems, and decision support systems among others (see [4,21,26,30]). Intuitively, an argument is a coherent set of statements that supports a claim; the acceptance of this claim will depend on a dialectical analysis (formalized through an inference procedure) of the arguments in support of the claim and considering those arguments against that support [30].

In the literature, a particular kind of argumentation systems can be identified, namely *Rule Based Argumentation Systems* (RBAS) [2,14,18,28]. In RBAS, arguments are built from a specific knowledge

[*]Corresponding author. E-mail: cadd@cs.uns.edu.ar.

base of rules and facts, which in many cases represents an important drawback, as all the available information needs to be explicitly encoded in such rules and facts; this may be an obstacle for their application to real world situations where often there is an enormous amount of data to be included, and encoding explicitly such data would make the RBAS highly impractical [10]. Because of this problem, recently there has been an increasing interest in connecting argumentation approaches with massive data repositories, such as relational databases or the Web [7,13,22,29,36].

The research presented here is related to those works focused on enabling argumentation over massive repositories but following a different direction. Our goals are not only to provide means of interaction between the argumentation mechanisms and relational databases, but also to introduce means of *controlling* the interaction itself and how it affects the proof procedure used to warrant claims through the argumentation process. To achieve this, we will take as the basis for our work the DBI-DeLP (Database Integration for Defeasible Logic Programming) framework developed in [13], revising and expanding several parts in it to achieve the mentioned goals. In particular, a key aspect to be considered for this is that different information sources can have attached different epistemic importance (whether it reflects credibility, weight, probability, etc.); thus, it is important which particular source is providing the support for literals in arguments. Moreover, such epistemic importance may vary as the topic changes, and then we argue that it is not advisable to simply establish an order among databases, but rather we have also to consider literals themselves in the equation.

To model this, we will introduce the notion of *authority*, which represents the topic expertise associated with a specific database regarding the particular predicate it gives support to; using this concept we introduce eDBI-DeLP (extended DBI-DeLP), an extension of framework developed in [13], whose main contribution will be twofold. On the one hand, we introduce a mechanism to control how the data supporting an argument is to be retrieved from databases; this is achieved by using *filters*, which provide the user with a way to restrict the databases in which the data should be searched. These filters can take different forms; for instance, we can restrict the search to those databases with a maximal authority for the given predicate, or ask to retrieve those results which have more consensus on the set of available databases. On the other hand, we provide an argument comparison criterion tailored for our scenario of obtaining data to build arguments from databases. Such comparison criterion will be based on the use of an argument valuation function that considers both the inherent strength associated with rules in the program and the authority assigned to the databases that provide the data that enables the use of these rules. In this way, the dialectical process outcome becomes influenced by the authority of the repositories used to construct arguments, a feature which can be of great benefit in different application domains by focusing the process on the pertinent data. For instance, in a medical setting we can have databases corresponding to different hospitals, attaching higher authority to those pieces of information coming from hospitals specialized in various areas of expertise (*e.g.*, burn and skin injuries, cancer treatments, etc.), rather than plainly adopt a general ranking among all hospitals. This approach can help to identify which arguments should prevail in different situations (*e.g.*, a good argument concerning how to treat skin cancer should combine high-quality information from the skin-specialized hospital and the cancer-specialized hospital).

The remainder of the paper is organized as follow: in Section 2, we review DeLP, the formalism that supports eDBI-DeLP; in Section 3, we outline a possible structure that allows the realization of argumentation processes over information stored in databases, focusing on how to control the interaction between the rules in the program and relational databases, and how such interaction influences the dialectical process used to warrant arguments; in Section 4, we present a complete structure that enables the objectives of this paper. Finally, in Section 5, we introduce the main conclusions obtained and discuss the pertinent related works, identifying lines for future research work as well.

2. Preliminaries

We will begin by giving a brief summary of Defeasible Logic Programming (DeLP) [18], a formalism that combines results from Logic Programming and Defeasible Argumentation providing the possibility of representing information as rules in a declarative manner, and a defeasible argumentation inference mechanism for warranting the entailed conclusions. These rules are the key element for introducing defeasibility and they are used to represent a relation between pieces of knowledge that could be defeated after all things are considered.

Assuming certain familiarity with Logic Programming, the language of DeLP can be succinctly described as follows. A term is either (a) a constant, (b) a variable, or (c) if f is a functor and $t_1, t_2, \ldots, t_n$ are terms then $f(t_1, t_2, \ldots, t_n)$ is a term. Also, if p is a predicate and $t_1, t_2, \ldots, t_n$ are terms then $p(t_1, t_2, \ldots, t_n)$ is an atom, and a literal is an atom or a strongly negated atom (an atom preceded by the strong negation "$\sim$"). A *defeasible logic program* (or delp for short) is a pair (Π, Δ) where Π is a set of strict rules and facts, and Δ is a set of defeasible rules. In a delp, knowledge can be represented using strict rules, facts, and defeasible rules. Facts are ground literals representing atomic information or the negation of atomic information using strong negation "$\sim$", *e.g.*, a or $\sim a$. DeLP strict rules are denoted $L_0 \leftarrow L_1, \ldots, L_n$, and represent information that cannot be attacked, *i.e.*, if the *Body* consisting of $L_1, \ldots, L_n$ can be supported then the *Head* L_0 it is also supported; while defeasible rules (d-rules) are denoted $L_0 \prec L_1, \ldots, L_n$ which represents tentative information that may be used if nothing could be posed against it. A d-rule *Head* $\prec$ *Body* expresses that *reasons to believe in Body give reasons to believe in Head*.

Strict and defeasible rules are ground, nevertheless, following the usual practice in logic programming [23], we use schematic rules with (meta-)variables in them, that stand for all possible grounded instances of such rules. To distinguish these variables from other elements in a schematic rule, we adopt the notation of logic programming, where variable names begin with uppercase letters, and where constant and predicate names begin with lowercase letters. For example, actor(Person) $\leftarrow$ performs_in(Movie, Person) represents a strict rule; and good_movie(Movie) $\prec$ performs_in(Movie, arnold) represents a defeasible rule.

A defeasible rule with an empty body is called a *presumption*. Presumptions are assumed to be true if nothing could be posed against them. In [18,25] an extension to DeLP that includes presumptions is presented, where an *extended delp* is a set of facts, strict rules, defeasible rules and presumptions.

From a DeLP program it is possible to infer tentative information and these inferences are called defeasible derivations, and are computed by backward chaining applying the usual SLD inference procedure used in logic programming. Strong negation can appear in facts and presumptions, or generally in the head and body of strict and defeasible rules; therefore, it is important to note that from a program it is possible to obtain contradictory literals; but, the set Π used to represent non-defeasible information is non-contradictory, *i.e.*, Π is such that no pair of contradictory literals can be derived from Π. This last restriction is methodological since from an inconsistent Π all the language can be obtained.

An *argument* for a literal L, denoted $\langle \mathcal{A}, L \rangle$, is a minimal non-contradictory set of d-rules $\mathcal{A} \subseteq \Delta$, that allows to derive L, possibly in combination with the strict information available. When contradictory literals are derived from a program, a dialectical process is used to decide which literals are *warranted*;

that is, a literal L is warranted if there exists a non-defeated argument $\mathcal{A}$ for L. To establish if $\langle \mathcal{A}, L \rangle$ is a non-defeated argument, defeaters for $\langle \mathcal{A}, L \rangle$ are considered. Counterarguments of $\langle \mathcal{A}, L \rangle$ are those arguments that disagree (*i.e.*, are in contradiction) at some point with $\langle \mathcal{A}, L \rangle$, and a counter-argument is a defeater for an argument $\mathcal{A}$ if it is preferred to $\langle \mathcal{A}, L \rangle$ by some argument comparison criterion. Note that different comparison criteria can be formulated to suit the needs of a particular application domain; in particular, in this work we will focus on comparison criteria which take into account the characteristics of the databases used as source of information. More detailed information regarding how the warrant procedure is performed can be found in [18,19].

It seems natural that in a setting where arguments are supported by information extracted from databases, the characteristics of such databases, *e.g.*, the credibility associated with them should influence the preference of an argument over another. Moreover, we argue that those characteristics should take into account the topics considered as well, since a database could be reliable for a particular topic but be completely inadequate for another; for instance, a database with movie information provided by users can be a good resource when we try to find out by looking at the ratings provided by the users whether a movie is good or not, but it may not be reliable when trying to find out the salaries earned by the actors who participated in the film and, on the other hand, a database provided by the Internal Revenue Service should be a good fit for the latter topic, but not for the former. To model this, we introduce the concept of *authority*, denoted $\mathcal{V}$, which will have a value inside $[0, 1]$ representing how well fitted a database is at supporting some literal. In the following, we assume the existence of a set of authorities for the literals that appear in the rules of a program. So, for instance, we can have a predicate used to assess an actor's performance in movies and databases DB_{critics} and DB_{users}, and we can set the authority of the former database to 1 while the latter's authority to 0.5 (deeming the critics' valuation of performances as more significant than those from regular moviegoers). Later in the paper, we will see how these values will be considered in the dialectical process when evaluating arguments, thus affecting the outcome of the argumentation process.

3. Defeasible argumentation over databases

DeLP enables query resolution by an argumentative process which deals with incomplete and potentially contradictory information. Several real-world applications have been proposed on the basis of DeLP, such as recommender and decision-support systems [27], multi-agent systems [35], agreement technologies [6], *etc*. However, many of these real-world environments require massive repositories of data; thus, DeLP requires additional features to handle such data, since this cannot be achieved by including new data as a static part of the program. The DBI-DeLP framework [13] is an extension of DeLP that uses relational databases as the source of the information on which arguments are based. This framework has been proven useful in the definition of new architectures for argumentation-based applications which have the above mentioned requirements [9]. In what follows, we will add new components to the framework presented in [13], and redefine others in such a way they suit our objectives. The extended DBI-DeLP framework will be called eDBI-DeLP.

3.1. Representing knowledge in the framework: Interaction between rules and databases

The eDBI-DeLP framework will enable argumentation supported by information stored in relational databases. In what follows, we will describe how this can be achieved, introducing ways of representing the information stored in the databases in the context of a DeLP-like program. First, we will introduce an annotated form of literals extending the notation with additional information regarding the information the literals represent; this extra information may take diverse forms, *e.g.*, describe conditions that must be met for the literal to hold, or provide information about which sources we shall use to support the literals. Then, an *annotated literal* is defined as follows:

Definition 1 (Annotated literal)**.** Given an atom $pred(p_1, \dots, p_m)$ and an atom *filter*, then "$pred(p_1, \dots, p_m)[filter]$" is an *annotated literal.*

The extra information is called *filters* and the idea behind them is to give the programmer more control on the interactions between the program and the databases, by augmenting the expressiveness of the former. Filters can be used for diverse purposes and later we will formally define the filters that will be used in our framework, but for now we will introduce the intuitions behind the set of filters that will used here.

- ϵ filter: indicates that the literal is supported from databases; thus, if a literal with this filter is included in a rule it is unnecessary to look into the facts in the program to prove the literal.
- *max* filter: indicates that we should retrieve only tuples from databases that have the maximal authority value for the predicate that the filter is annotating; for instance, $weather(london, Weather)[max]$ establish to only use information from the databases with the highest authority for the predicate $weather(City, Weather)$.
- *maj* filter: states that we should use a majority approach when retrieving information. That is, we should retrieve information that is replicated in most databases; for instance, $weather(london, Weather)[maj]$ indicates to retrieve the weather value that appears in the most available databases containing the weather associated with the city London.
- *source* filter: establishes which databases we will search for support for the annotated literal; *e.g.*, $rating(Movie, Rating)[source(DB_{\text{critics}}, DB_{\text{directors}})]$ is an indication that we should retrieve the ratings of movies only from the DB_{critics} and the $DB_{\text{directors}}$ databases. Notice that this filter is used to narrow the search space; thus, if some annotated literal does not have a *source* filter then the search is performed in every available database suitable for supporting the literal.
- *excluded_source* filter: prohibits some source to be used as support for the literal; for instance, $movie\_rating(Movie, Rating)[excluded\_source(DB_{\text{users}})]$ ignores the Users' database when retrieving information for such literal.

Then, filters can be used by a developer to control how the rules interact with the databases that provide literals used to support arguments. To do this, in this work we redefine strict and defeasible rules to allow the use of annotated literals in their body.

Definition 2 (Strict rules and defeasible rules)**.** Given a literal L, *i.e.*, a ground atom or a negated ground atom, and a finite, non empty, set $Body = \{L_1, \dots, L_n\}$, where all $L_i (1 \leqslant i \leqslant n)$ are literals or annotated literals:

- A *strict rule* is an ordered pair "$L \leftarrow Body$".
- A *defeasible rule* is an ordered pair "$L \prec Body$".

To properly represent information coming from relational databases the eDBI-DeLP framework adopts the notion of *presumptions* [18] for representing "defeasible" facts, thus avoiding inconsistencies in the strict knowledge as required in DeLP. Given a database D, their *operative presumptions* (OPs) are those tentative facts associated with the information stored in D. Such tentative facts are represented by means of annotated literals, and then they contain information of the filters used in the retrieval of tuples from databases (if any). In addition, in this work we will attach to every presumption which is supported by tuples in a relational database (*i.e.*, to every OP) additional information regarding the authority of the sources that gives support to the operative presumption. Formally, these presumptions are defined as follows:

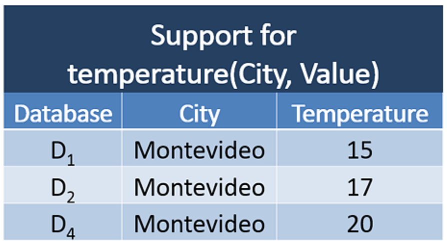
Definition 3 (Annotated operative presumption)**.** Let $\mathbf{X}$ be a set of predicates, *pred* the predicate name for some predicate $x \in \mathbf{X}$, and $\mathbf{D} = \{D_1, \dots, D_n\}$ a set of databases. An *annotated operative presumption* (OP) for a database D and the predicate *pred* is a pair $(AL, \mathcal{V})$ where:

- AL is an annotated literal with the form $pred(t_1, \dots, t_m)[filter]$ such that there exists a tuple $tup = (q_1, \dots, q_m) \in D$, where $D \in \mathbf{D}$ and $q_i = t_i$ for all i, and
- $\mathcal{V} \in [0, 1]$

Given an operative presumption $O = (pred(t_1, \dots, t_m)[filter], \mathcal{V})$ we denote by $lit(O)$ the literal part of the operative presumption, *i.e.*, $lit(O) = pred(t_1, \dots, t_m)$.

The set of all OPs for given sets of predicates X and set of databases $\mathbf{D} = \{D_1, \dots, D_n\}$, denoted as $OPset_{\mathbf{X},\mathbf{D}}$ is defined as $\bigcup_{i=1}^{n} OPset_{\mathbf{X},D_i}$, where $OPset_{\mathbf{X},D_i}$ is the set of all OPs for database D_i and every predicate $x \in \mathbf{X}$.

Example 1. Suppose that we have the predicate $temperature(City, Value)$, and the databases D_1, D_2 and D_4 that have information regarding temperatures in different cities, where the authority assigned to D_1 is 0.7, and the authority given to D_2 and D_4 is 0.5. Now, assume that for the city Montevideo we have the following information



Support for temperature(City, Value)		
Database	City	Temperature
D_1	Montevideo	15
D_2	Montevideo	17
D_4	Montevideo	20

Then, we can build the OP $(temperature(montevideo, 17)[\epsilon], 0.5)$.

To see how filters affect the way OPs are built consider now the case where we have $temperature(City, Value)[max]$. Then, the previous OP cannot be built. Nevertheless, we can have $(temperature(montevideo, 15)[max], 0.7)$.

In this manner, operative presumptions are used to represent the information available in the databases so they could be used to build arguments. Nevertheless, given that in this work we use annotated operative presumptions which are different from the presumptions used in [18], then we could not simply reuse the standard derivation in DeLP used in that work. Then, we present now a modified version of derivation, tailored to our particular setting.

Definition 4 (DB-based defeasible derivation)**.** Let Π be a set of strict rules and facts, Δ be a set of defeasible rules, $\mathbf{X}$ the set of every predicate in the rules of $\Pi \cup \Delta$, $\mathbf{D} = \{D_1, \dots, D_n\}$ a set of databases, $OPs \subseteq OPset_{\mathbf{X},\mathbf{D}}$ and L a ground literal. A defeasible derivation of L from $\mathcal{P} = \Pi \cup \Delta \cup OPs$, denoted $\mathcal{P} \mid\!\sim L$, consists of a non-empty finite sequence $L_1, L_2, \dots, L_n = L$ of ground literals, and each literal L_i is in the sequence because:

(a) $L_i \in \Pi$ is a fact, or
(b) $(AL, \mathcal{V}) \in OPs$ is such that $lit(AL) = L_i$, or
(c) there exists a rule R_i in $\mathcal{P}$ (strict or defeasible) with head L_i and body B such that:

- if $B_j \in B$ is a literal then B_j is an element of the sequence appearing before L_i.
- if $B_j \in B$ is an Annotated Literal then there is an operative presumption $OP = (B_j, \mathcal{V})$ in OPs such that $lit(B_j)$ is an element of the sequence appearing before L_i.

When there is some operative presumption or defeasible rule in a derivation we call such derivation a *defeasible derivation*.

As a remark note that here we use the same notational practice adopted from Programming Logic for DeLP regarding schematic rules. That is, in eDBI-DeLP all strict and defeasible rules are grounded, but for notational simplicity we will present them using meta-variables that represent all the grounded instances of them. Nevertheless, if we were to use variables in the programs then we could apply a substitution schema such as the one introduced by Capobianco *et al.*, [10].

Arguments in eDBI-DeLP are obtained in the same way explained previously for DeLP, but using the derivation procedure stated in Definition 4.

Definition 5 (Arguments in eDBI-DeLP)**.** Let Π be a set of strict rules and facts, Δ be a set of defeasible rules, $\mathbf{X}$ the set of every predicate in the rules of $\Pi \cup \Delta$, $\mathbf{D} = \{D_1, \ldots, D_n\}$ a set of databases and L a ground literal. A pair $\langle \mathcal{A}, Q \rangle$ where $\mathcal{A}$ is a non empty set $\mathcal{A}$ of defeasible rules in Δ and operative presumptions in $OPset_{\mathbf{X},\mathbf{D}}$ and Q is a grounded literal is an argument for Q iff:

(1) there exists a defeasible derivation for Q from $\Pi \cup \mathcal{A}$,
(2) $\Pi \cup \mathcal{A}$ is non contradictory,
(3) $\mathcal{A}$ is minimal with respect to set inclusion in satisfying (1) and (2).

Given an argument $\langle \mathcal{A}, Q \rangle$ we call the literal Q the claim of $\mathcal{A}$, denoted $claim(\mathcal{A})$. The set of all arguments that can be built from $\mathcal{P} = \Pi \cup \Delta \cup \mathbf{D}$ is denoted $\mathcal{A}rgs$.

For the sake of simplicity through the rest of the paper we will often omit the claim in an argument, *i.e.*, we will refer to $\langle \mathcal{A}, Q \rangle$ as $\mathcal{A}$.

An eDBI-DeLP program (*edbi-delp*) accounts for a DeLP program (as defined in Section 2) along with a set Σ of operative presumptions, associated with a number of available databases $D_1, \ldots, D_k$. Such operative presumptions are built on demand for solving a query, and discarded later when the query has been solved. Finally, an *edbi-delp* contains a valuation υ among the facts, strict and defeasible rules in the program, which can be used to establish priorities between such elements. We assume that this valuation is such that strict elements (both facts and strict rules) have a value of 1, whereas defeasible rules' valuation is a real in [0, 1]. Note that the choice of strict information having a value of 1 is made to model that strict information cannot be challenged, and thus it has the maximal valuation possible. As we will see later in the paper such valuation has an impact on the overall dialectical process, adding an extra feature to ultimately determine whether a claim is accepted or not.

Definition 6 (eDBI-DeLP program)**.** Let $\mathbf{D} = \{D_1, \ldots, D_k\}$ be a set of databases, an *eDBI-DeLP program* (*edbi-delp*) $\mathcal{P}$ is a tuple $(\Pi, \Delta, \mathbf{D}, \Sigma, \upsilon)$ where Π is a set of facts and strict rules, Δ is a set of defeasible rules, $\Sigma = OPset_{\mathbf{X},\mathbf{D}}$ is a set of Operative Presumptions where $\mathbf{X}$ is the set of every predicate in the rules of (Π, Δ), and $\upsilon : \Pi \cup \Delta \mapsto [0, 1] \in \mathbb{R}$.

Now we describe the process used to answer queries from an *edbi-delp*. In Definition 4 we have outlined how eDBI-DeLP constructs arguments to solve queries by a backward chaining process. That is, when eDBI-DeLP is searching for an argument in support of a literal L, the argument construction might involve a strict or defeasible rule having L in the head; then, DeLP tries to prove the literals in the body of this rule. These literals in the body are called *Target Goals* (*TG*), as they will be the next *goals* of the inference procedure.

Definition 7 (Target goals)**.** Let Π be a set of facts and strict rules, and Δ a set of defeasible rules. Given some strict rule $L \leftarrow L_1, \ldots, L_n$ in Π or defeasible rule $L \prec L_1, \ldots, L_n$; every (annotated) literal L_i, $(1 \leqslant i \leqslant n)$, in the body of the rule is called Target Goal (*TG*). The set of all TGs for $\Pi \cup \Delta$ is called *TGset*.

The *TG*s are a key element in eDBI-DeLP, as they are the connection between the rules in the program and the records in databases. As such, since they emerge from rules then TGs clearly depend on the particular sets of strict and defeasible rules being considereng in an *edbi-delp*. Every TG will be analyzed following the traditional SLD procedure, using all the rules, facts and presumptions in the *edbi-delp*. For the purpose of this work, we focus on how presumptions can be obtained from the available databases. To do so, a search for operative presumptions is launched to retrieve from the databases information offering support to the literal (if any). For this, we begin by identifying the data sources; *i.e.*, the databases, and the tables and fields in it, that are expected to have useful data for the *TG*. The triplet [*database, table, field*] in the data source is called a Parameter Source (*PS*), formally,

Definition 8 (Parameter source)**.** Given a set $\mathbf{D} = \{D_1, \ldots, D_n\}$ of available databases, a PS is a triplet $[D_i, T, F]$ where $D_i \in \mathbf{D}$, T is a table in D_i and F is a field in T. The set of all PS for a given $\mathbf{D}$ is called *PSS*.

Each potential data source of useful information for a given *TG* is linked to the corresponding *TG* through a Pertinence Relation. Moreover, in this relation we find the information regarding the authority assigned to a database with respect to a particular predicate. It is important to remark that authority is information extra that is attached to the pertinence relation for further use in the warranting of arguments, but does not affect whether or not a data source is pertinent for a TG.

Definition 9 (Pertinence relation)**.** Given a set $\mathbf{D}$ of available databases $\{D_1, \ldots, D_n\}$, a set Π of facts and strict rules, and a set Δ of defeasible rules, let *PSS* be the set of all PS for $\mathbf{D}$*TGset* the set of all *TG* for $\Pi \cup \Delta$. The Pertinence Relation $PR \subseteq TGset \times 2^{PSS} \times [0, 1]$ is such that $(TG, DS, \mathcal{V}) \in PR$ iff $DS \subset PPS$ and $\mathcal{V} \in [0, 1]$.

We assume that the *Pertinence Relation* is given as an input to the system; in Section 4.3 we show how this relation is implemented through a particular structure.

Intuitively, if a data source is pertinent for a *TG* then we can use that data source to support that *TG*, *i.e.*, we can obtain the necessary tuples (and, in turn, built the necessary OPs) from this source. Once we know which data sources are pertinent, we have to retrieve from them the data and make it available to the DeLP core which builds answers to the query using this data, along with the rest of the *edbi-delp*. This retrieval is made by the application of the Presumption Retrieval Function (PRF).

Before introducing the PRF, however, we will introduce the Tuple To Presumption Function (TPF), as a previous step to formally define the filters that we will consider through the paper. As its name suggests, the TPF is a function that takes a set of tuples (possibly with some other parameters) and built presumptions that can be used by the argumentative process presented in the paper. Formally, a TPF is as follows.

Definition 10 (Tuple to presumption function)**.** Let $\mathbf{D} = \{D_1, \dots, D_n\}$ be a set of available databases, *TGset* be the set of all target goals and *OPset* be the set of all operative presumptions for $\mathbf{D}$ and *TGset*. Let $TG \in TGset$, and finally, let $TupSet = \{(tup_1, D_i), (tup_2, D_{i'}), \dots, (tup_k, D_i''))\}$ where $1 \leqslant i, i', i'' \leqslant n$ be a set of pairs such that (tup, D_i) indicates that $tup \in D_i$.

A Tuple To Presumption Function *TPF* is such that if $TPF(TupSet, TG) = S$ then the set $S \subseteq OPset$ is a set of Operative Presumptions.

Clearly, the TPF models a family of functions that transform tuples (along with additional information) into operative presumptions, rather than a particular constructive function. In this work we call *filters* to the particular instances of the TPF that we will use when operative presumptions are built.

So, filters are functions that focus on certain subsets of the whole set of presumptions that could be retrieved from the databases to support certain literals. We have already intuitively introduced the set of filters that we will use in this work. Now we will look more deeply into how such filters behave, that is, we will introduce the particularizations of the TPF that we will use through this paper. In what follows let $\mathbf{D} = \{D_1, \dots, D_n\}$ be a set of available databases, *TGset* be the set of all target goals and *OPset* be the set of all operative presumptions. Let $TG \in TGset$, *PSS* be the set of all parameter sources, and *PR* be the Pertinence Relation for *TGset* over *PSS*. Given a tuple $tup = (q_1, \dots, q_m)$, let $|tup_{\mathbf{D}}|$ be the number of databases $D_i \in \mathbf{D}$ such that $tup \in D_i$. Finally, let $TupSet = \{(tup_1, D_i), (tup_2, D_{i'}), \dots, (tup_k, D_i''))\}$ where $1 \leqslant i, i', i'' \leqslant n$ be a set of pairs such that (tup, D_i) indicates that $tup \in D_i$, and S be the set of Operative Presumptions returned by the filter. Then the filters used in this work are the following.

The ϵ filter simply builds the presumptions based on the tuples received and the authority for the pair (TG, DB), with no further filtering being done.

Definition 11 (ϵ filter)**.** Given $TG = pred(t_1, \dots, t_m)$ or $TG = pred(t_1, \dots, t_m)[\epsilon]$, the ϵ filter (or empty filter) is such that $(pred(t_1, \dots, t_m)[\epsilon], \mathcal{V})$ in $\epsilon(TupSet, TG)$ iff

- there exists $(tup, D) \in TupSet$ and $(TG, DS, \mathcal{V}) \in PR$ where $tup = (q_1, \dots, q_m)$ is such that for every $q_i \in tup$ it holds that q_i belongs to field F in table T of database D and $[D, T, F] \in DS$.

The max filter only returns those presumptions built based on tuples retrieved from databases with the maximal authority for the given TG.

Definition 12 (*max* filter)**.** Given $TG = pred(t_1, \dots, t_m)[max]$, the *max* filter is such that $(pred(t_1, \dots, t_m)[max], \mathcal{V})$ in $max(TupSet, TG)$ iff

- there exists $(tup, D) \in TupSet$ and $(TG, DS, \mathcal{V}) \in PR$ where $tup = (q_1, \dots, q_m)$ is such that for every $q_i \in tup$ it holds that q_i belongs to field F in table T of database D and $[D, T, F] \in DS$, and
- there does not exist $DS' \in PSS$ where $(TG, DS', \mathcal{V}') \in PR$ such that $\mathcal{V} < \mathcal{V}'$.

The maj filter returns presumptions supported in the greater number of databases, assigning as the authority for the presumption the maximal authority among the ones assigned for the databases containing the tuple and the given target goal.

Definition 13 (*maj* filter)**.** Given $TG = pred(t_1, \dots, t_m)[maj]$, the *maj* filter is such that $(pred(t_1, \dots, t_m)[maj], \mathcal{V})$ in $maj(TupSet, TG)$ iff

- there exists $(tup, D) \in TupSet$ and $(TG, DS, \mathcal{V}) \in PR$ where $tup = (q_1, \ldots, q_m)$ is such that for every $q_i \in tup$ it holds that q_i belongs to field F in table T of database D and $[D, T, F] \in DS$ and there does not exist $(TG, DS', \mathcal{V}') \in PR$ such that for every $q_i \in tup$ it holds that q_i belongs to field F' in table T' of database D' and $[D', T', F'] \in DS'$ where $\mathcal{V} < \mathcal{V}'$, and
- there does not exist $DS' \in PSS$ where $(TG, DS', \mathcal{V}') \in PR$ and $tup' = (r_1, \ldots, r_m)$ such that for every $r_i \in tup'$ it holds that $r_i = t_i$, r_i belongs to field F' in table T' of database D' and $[D', T', F'] \in DS'$ and it holds that $|tup_{\mathbf{D}}| < |tup'_{\mathbf{D}}|$.

The source filter builds the presumptions based on the tuples received and the authority for the pair (TG, DB) under consideration, filtering out those based on tuples retrieved from databases that are not listed on its parameter.

Definition 14 (*sources* filter)**.** Given $TG = pred(t_1, \ldots, t_m)[sources(DBList)]$, the *sources* filter is such that $(pred(t_1, \ldots, t_m)[sources(DBList)], \mathcal{V})$ in $source(TupSet, TG)$ iff

- there exists $(tup, D) \in TupSet$ and $(TG, DS, \mathcal{V}) \in PR$ where $tup = (q_1, \ldots, q_m)$ is such that for every $q_i \in tup$ it holds that q_i belongs to field F in table T of database D and $[D, T, F] \in DS$, and $D \in DBList$.

The excluded_source filter builds the presumptions based on the tuples received and the authority for the pair (TG, DB) under consideration, filtering out those based on tuples retrieved from databases listed on its parameter.

Definition 15 (*excluded_sources* filter)**.** Given $TG = pred(t_1, \ldots, t_m)[excluded\_sources(DBList)]$, the *excluded_sources* filter is such that $(pred(t_1, \ldots, t_m)[excluded\_sources(DBList)], \mathcal{V})$ in *excluded_ source*$(TupSet, TG)$ iff

- there exists $(tup, D) \in TupSet$ and $(TG, DS, \mathcal{V}) \in PR$ where $tup = (q_1, \ldots, q_m)$ is such that for every $q_i \in tup$ it holds that q_i belongs to field F in table T of database D and $[D, T, F] \in DS$, and $D \notin DBList$.

The set of filters introduced is the particular one that we will consider in the present work, but clearly could be expanded to suit other application environment needs if necessary. We can show that the filters *max*, *maj*, *source* and *excluded_source* are proper instances of TPF (that is, the ϵ filter).

Proposition 1. *Let* $TupSet = \{(tup_1, D_i), (tup_2, D_{i'}), \ldots, (tup_k, D''_i)\}$ *where* $1 \leqslant i, i', i'' \leqslant n$ *be an arbitrary set of pairs such that* (tup, D_i) *indicates that* $tup \in D_i$*, and TG be an arbitrary Target Goal.*

Then, for any $OP \in filter(TupSet, TG)$ *there exists* $OP' \in \epsilon(TupSet, TG)$ *such that* $lit(OP) = lit(OP')$*, where* $filter \in \{max, maj, sources(DBList), excluded\_sources(DBList')\}$.

Proof. Let $TupSet = \{(tup_1, D_i), (tup_2, D_{i'}), \ldots, (tup_k, D''_i)\}$ where $1 \leqslant i, i', i'' \leqslant n$ be an arbitrary set of pairs such that (tup, D_i) indicates that $tup \in D_i$, and TG be an arbitrary Target Goal.

We begin with the analysis of the *max* filter. Consider any arbitrary operative presumption $OP = pred(t_1, \ldots, t_m)[max]$ such that $OP \in max(TupSet, TG)$. Then, by Definition 12 there exists $(tup, D) \in TupSet$ and $(TG, DS, \mathcal{V}) \in PR$ where $tup = (q_1, \ldots, q_m)$ is such that for every $q_i \in tup$ it holds that q_i belongs to field F in table T of database D and $[D, T, F] \in DS(1)$.

From (1) and Definition 11 it follows that there exists $OP' \in \epsilon(TupSet, TG)$ such that $OP' = pred(t_1, \ldots, t_m)[\epsilon]$. Finally, we have that $lit(OP) = pred(t_1, \ldots, t_m)$ and that $lit(OP') = pred(t_1, \ldots, t_m)$, and thus since OP is arbitrary it holds that for any $OP \in max(TupSet, TG)$ there exists $OP' \in \epsilon(TupSet, TG)$ such that $lit(OP) = lit(OP')$.

We omit the proofs for filters *maj*, $sources(DBList)$ and $excluded\_sources(DBList')$ as they are analogous to the one presented for *max*. □

Now that we outlined all filters that will be considered in the paper, we are ready to introduce the Presumption Retrieval Function. The goal of the PRF is to feed the argumentation process with relevant data obtained from the pertinent data sources, along with information regarding the authority assigned to such datasources for the topic addressed by the particular literal that is trying to support.

Definition 16 (Presumption retrieval function)**.** Let $\mathbf{D} = \{D_1, \dots, D_n\}$ be a set of available databases, *TGset* be the set of all target goals and *OPset* be the set of all operative presumptions for $\mathbf{D}$ and *TGset*. Let $TG \in TGset$, *PSS* be the set of all parameter sources, and *PR* be the Pertinence Relation for *TGset* over *PSS*.

The Presumption Retrieval Function $PRF : TGset \mapsto 2^{OPset}$ is such that

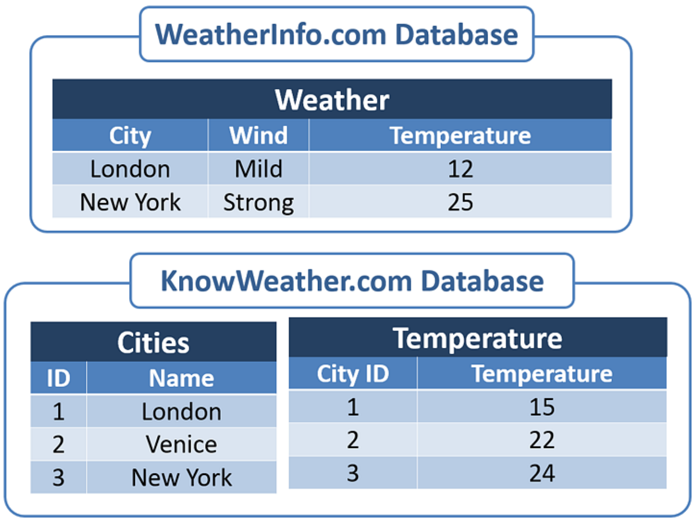
$$PRF(TG) = \begin{cases} \{OP \mid OP \in \epsilon(TupSet, TG)\} & \text{if } TG = pred(t_1, \dots, t_m) \text{or if} \\ & TG = pred(t_1, \dots, t_m)[\epsilon], \\ & \textit{i.e.}, TG \text{ is a literal or a} \\ & \text{non-filtered annotated literal} \\ \{OP \mid OP \in filter(TupSet, TG)\} & \text{if } TG = pred(t_1, \dots, t_m)[filter] \\ & \text{where } filter \in \{max, maj, \\ & sources(DBList), \\ & excluded\_sources(DBList')\}, \\ & \textit{i.e.}, TG \text{ is a filtered annotated} \\ & \text{literal} \end{cases}$$

where $TupSet = \{(tup_1, D_i), (tup_2, D_{i'}), \dots, (tup_k, D_i'')\}$ is such that

- for every $(tup, D) \in TupSet$ it holds that $tup = (q_1, \dots, q_m)$ exists in the database $D \in \mathbf{D}$ such that $q_i = t_i$, for all $1 \leqslant i \leqslant n$, and
- there exists $(TG, DS, \mathcal{V}) \in PR$ such that for every $q_i \in tup$ it holds that q_i belongs to field F in table T of database D and $[D, T, F] \in DS$.

Therefore, the PRF function retrieves database tuples from pertinent data sources with values equal to the corresponding constant values. Notice how the PRF function formalizes the effects of the different filters available by conditioning the retrieval of tuples. Clearly, Definition 16 is flexible enough to allow an easy expansion of the set of filters proposed.

To see an example of how the PRF works, consider a database DB_{actors} with the tuples (demolition man, stallone), (demolition man, snipes), and (rambo, stallone), where each tuple states that the actor in the second component has appeared in the film associated with the first component. Given the *TG* performs_in(demolition man, Actor), assume that the authority of database DB_{actors} for the predicate performs_in is 0.9. Then, for the PRF we have


WeatherInfo.com Database

Weather

City	Wind	Temperature
London	Mild	12
New York	Strong	25

KnowWeather.com Database

Cities

ID	Name
1	London
2	Venice
3	New York

Temperature

City ID	Temperature
1	15
2	22
3	24

Fig. 1. Weather databases.

$$\begin{aligned}\text{PRF}(\text{performs\_in(demolition man, Actor)}) = \\ \{(\text{performs\_in(demolition man, stallone)}, 0.9) \\ (\text{performs\_in(demolition man, snipes)}, 0.9)\ \}\end{aligned}$$

unifying the non grounded parameter (*i.e.*, schematic variable) Actor with both stallone and snipes with an assigned authority of 0.9. As we can see, (performs_in(rambo, stallone), 0.9) is not in S as the tuple (rambo, stallone) does not match the value required for the grounded parameter.

Now, to see an example of how filters are applied consider the following situation. Suppose that we have the databases depicted in Fig. 1, and the *TG* $temperature(City, Temp)[max]$, where the authorities assigned for the $temperature(City, Temp)$ predicate for the WeatherInfo and KnowWeather databases is 0.65 and 0.25, respectively. Then, we have that:

$$\begin{aligned}\text{PRF}(temperature(City, Temp)[max]) = \\ \{(temperature(london, 12)[max], 0.65), \\ (temperature(new\_york, 25)[max], 0.65), \\ (temperature(venice, 22)[max], 0.25)\}\end{aligned}$$

3.2. *Argument preference in database-supported argumentation processes*

So far we have introduced a retrieval mechanism to obtain data to support the construction of arguments from databases together with other information associated that can be used to measure the value of such information. As explained in Section 2, to find out whether a literal is warranted, a dialectical process is carried out, looking to find an argument for that literal that is not defeated under an established defeat relation. Even when it is possible to abstract from the way in which the defeat relation is obtained, usually in practice it is better to be able to calculate the relation by taking advantage of the particular meaning of the domain [17]. A key component to obtain a defeat relation is the argument comparison criterion, which is used to decide whether or not an argument is preferred to another. In this section we will introduce a comparison criterion that we will use in eDBI-DeLP. Such comparison criterion makes full use of the authority assigned to a database regarding its specific topic (*i.e.*, the different predicates) to model topic expertise.

Next we define the rule in an argument that is used to infer the corresponding claim supported by that argument. Formally:

Definition 17 (Claim inferring rule)**.** Let $\mathcal{P} = (\Pi, \Delta, \mathbf{D}, \Sigma, \upsilon)$ be an *edbi-delp*, and an argument $\mathcal{A} \in \mathcal{A}rgs$ built from $\mathcal{P}$ such that $claim(\mathcal{A}) = L$ and δ its associated derivation. Then, the *claim inferring rule* $r \in \Pi \cup \Delta$ for argument $\mathcal{A}$ is such that either

- $r \in \mathcal{A}$ is a defeasible rule with $Head(r) = L$, or
- r is a strict rule with $Head(r) = L$, every literal $L' \in Body(r)$ is such that L' in δ and there exists no strict rule $r' \in \Pi$ such that $Body(r') \subsetneq Body(r)$.

We will now establish how the value of an argument is obtained in eDBI-DeLP, *i.e.*, how the authority assigned to presumptions and the valuation in rules is combined to calculate a final value for an argument. Formally, the argument valuation in eDBI-DeLP proceeds as follows.

Definition 18 (Argument valuation function)**.** Given an *edbi-delp* $\mathcal{P} = (\Pi, \Delta, \mathbf{D}, \Sigma, \upsilon)$, let $\mathcal{A}$ be an argument for L over $\mathcal{P}$. Then, the argument valuation function $\nu : \mathcal{A}rgs \mapsto [0, 1] \in \mathbb{R}$, noted as $\nu(\mathcal{A})$, is such that:

$$\nu(\mathcal{A}) = \begin{cases} \mathcal{V} & \text{if } \mathcal{A} = \{(AL, \mathcal{V})\} \\ 1 & \text{if } \mathcal{A} = \emptyset \\ \mu * \nu(\mathcal{A}') & \text{otherwise, where } (r, \mu) \in \upsilon \text{ and } r \text{ is the claim inference} \\ & \text{rule for } \mathcal{A}, \text{ and } \mathcal{A}' \subset \mathcal{A} \text{ is the subargument with minimum} \\ & \text{valuation such that } claim(\mathcal{A}') \in Body(r). \end{cases}$$

Intuitively, the valuation of an argument is made by combining the intrinsic valuation of its claiming rule with the valuations of its subarguments. To do so, we take the product of the less-valued element in the body of the rule and the numerical valuation of the rule, following a cautious approach similar to the propagation of necessity degrees in P-DeLP (Possibilistic Defeasible Logic Programming) [1]. Nevertheless, it is clear that this approach can be modified as needed to suit the particular requirements of the different application environments, *e.g.*, we could use an average of the values obtained for the elements in the body.

In eDBI-DeLP the argument valuation function will be used as the argument comparison criterion. First, however, we will formally introduce the notion of counter-argument. Intuitively, an argument counter-argues another when a contradiction arise when considering both arguments along with the strict information in the program. Formally this is as follows.

Definition 19 (Counterargument)**.** Given $\mathcal{P} = (\Pi, \Delta, \mathbf{D}, \Sigma, \upsilon)$, let $\langle \mathcal{A}, Q_1 \rangle$ and $\langle \mathcal{B}, Q_2 \rangle \in \mathcal{A}rgs$ be two arguments over $\mathcal{P}$. We say that $\langle \mathcal{A}, Q_1 \rangle$ is a *counter-argument* for $\langle \mathcal{B}, Q_2 \rangle$, if and only if there exists a subargument $\langle \mathcal{B}', Q \rangle$ of $\langle \mathcal{B}, Q_2 \rangle$ such that Q and Q_1 *disagree* (*i.e.*, $\Pi \cup \{Q, Q_1\}$ is contradictory).

Therefore, using this notion of counter-argument and the argument valuation in eDBI-DeLP, we will formalize the notion of defeater in our framework.

Definition 20 (Defeat). Given $\mathcal{P} = (\Pi, \Delta, \mathbf{D}, \Sigma, \upsilon)$, let $\langle \mathcal{A}, Q_1 \rangle, \langle \mathcal{B}, Q_2 \rangle \in \mathcal{A}rgs$ be two arguments over $\mathcal{P}$. Then, $\langle \mathcal{A}, Q_1 \rangle$ defeats $\langle \mathcal{B}, Q_2 \rangle$ iff $\langle \mathcal{A}, Q_1 \rangle$ is a counter-argument for $\langle \mathcal{B}, Q_2 \rangle$ attacking at literal Q, there exists a subargument $\langle \mathcal{B}', Q \rangle$ of $\mathcal{B}$ and

(a) $\nu(\mathcal{B}') < \nu(\mathcal{A})$ (proper defeater), or
(b) $\nu(\mathcal{B}') = \nu(\mathcal{A})$ (blocking defeater).

Example 2. To see how the argument valuation and the comparison criterion work, let's consider the following situation. Suppose that in our program we have the rules

$$sr_1 = \quad \sim buy\_stock(Company) \leftarrow good\_price(Company), risky\_company(Company)$$

and

$$dr_2 = \quad buy\_stock(Company) \prec good\_price(Company), in\_fusion(Company, AnotherCompany), value(AnotherCompany, Val)[max], worth(Val).$$

where the valuations for the rules are $\upsilon(sr_1) = 1$ and $\upsilon(dr_2) = 0.6$.

Now, assume that we have the argument:

$$\mathcal{A} = \{(good\_price(acme), 0.8), (risky\_company(acme), 0.25)\}$$

which uses sr_1 to conclude $\sim buy\_stock(acme)$, advising not to buy stocks from the company Acme, since it is a risky company. For the sake of example assume that we obtain the support for Acme being at a good price at this moment from a database with an authority of 0.8 for the topic, and that the literal *risky_company(acme)* is supported by a database that has an associated authority for that literal of 0.25, thus enabling us to build the operative presumption $(risky\_company(acme), 0.25)$. Under such assumptions, the value of argument $\mathcal{A}$ is $\nu(\mathcal{A}) = 1 * min(0.8, 0.25) = 1 * 0.25 = 0.25$.

On the other hand, assume that we build argument $\mathcal{B}$ as follows:

$$\mathcal{B} = \{(buy\_stock(acme) \prec good\_price(acme), in\_fusion(acme, emca), value(emca, 22M)[max], worth(22M)), (good\_price(acme), 0.8), (value(emca, 22M)[max], 0.7)\}$$

for *buy_stock(acme)*, where dr_2 is the claim inference rule, we have a fact stating Acme is merging with the company EMCA (which valuation is 1, since facts cannot be challenged), and the database with the maximal authority to inform company values between available databases (0.7) state that EMCA has a net value of 22 millions. Also, assume that we can build an argument that supports the claim that a company with a value $22M$ is worthy, and that the recursively obtained value for this argument is 1 (*e.g.*, only strict rules are used). Therefore, we have that $\nu(\mathcal{B}) = 0.6 * min(0.8, 1, 0.7, 1) = 0.6 * 0.7 = 0.42$.

In addition, note that $\mathcal{A}$ is a counter-argument for $\mathcal{B}$ and viceversa, since the literals $\sim buy\_stock(acme)$ and *buy_stock(acme)* disagree. Thus, we have that under our comparison criterion $\mathcal{B}$ defeats $\mathcal{A}$. Note that this occurs even when the claim inferring rule for $\mathcal{A}$ is a strict rule (which has maximum valuation) and the claim inferring rule for $\mathcal{B}$ is a defeasible rule (which has lower valuation). This is because the authority of the database for the information regarding acme being a risky company is considerably low.

4. Components of the framework

In order to integrate DeLP with a database system we need to identify which databases can be used during the argumentation process. We assume that our database system may involve several databases, which are accessed asynchronously. At runtime, new databases could be added or existing databases could be removed from the system. To formalize this setting in a seamless way, we must maintain compatibility with external systems, so that both the DeLP inference mechanism and the databases schemas can remain unchanged. To achieve this we establish a translation layer between the argumentation process and the databases, where the architecture is based on the following three components.

4.1. EDBI-DeLP server:

This component takes care of the argumentation process; that is, it receives a DeLP ground query, and then builds arguments and counter-arguments based on an *edbi-delp*, providing answers and explanations on how they were built. This component plays a central role in the framework, carrying out the valuation for rules and applying the comparison criterion for argument-based decision making. The component includes two modules to separate knowledge storage issues from the actual usage of such knowledge: Domain Logic and DeLP Core. Finally, the Rule Valuation Function is a component used to provide valuations for rules in the program.

– *Domain Logic*: is the knowledge of the domain that the system has. It is expressed as (part of) an eDBI-DeLP program. Thus, this component comprises strict rules and facts, defeasible rules, and also operative presumptions.
– *DeLP Core*: The argumentation process is carried out by the DeLP core. It receives a query from a client and tries to build arguments for and against it, and finally gives the obtained answer. Clearly, this is a key component in terms of the goal of our work, which is establishing how preference between arguments can be obtained by analyzing the different sources used. Note that this component will effectively apply the preference criteria for argument-based decision making. However, no major modifications need to be done to it, since the use of argument preference criteria in DeLP is modular [18], and thus we can directly change the comparison criterion, adapting it for its use when contrasting database-supported arguments.
– *Rule Valuation Function*: This component is used by the user to provide the υ relation in an *edbi-delp*; thus, it maintains the valuation for every rule in the Domain Logic.

4.2. Domain data Holder (DDH):

This component represents the set of databases that can be accessed to support arguments.

4.3. Domain data integrator (DDI):

The DDI retrieves the necessary information from the DDH and feeds it to the DeLP Core so it can be used in the argumentation process. The information needed by the DDI to perform this translation is

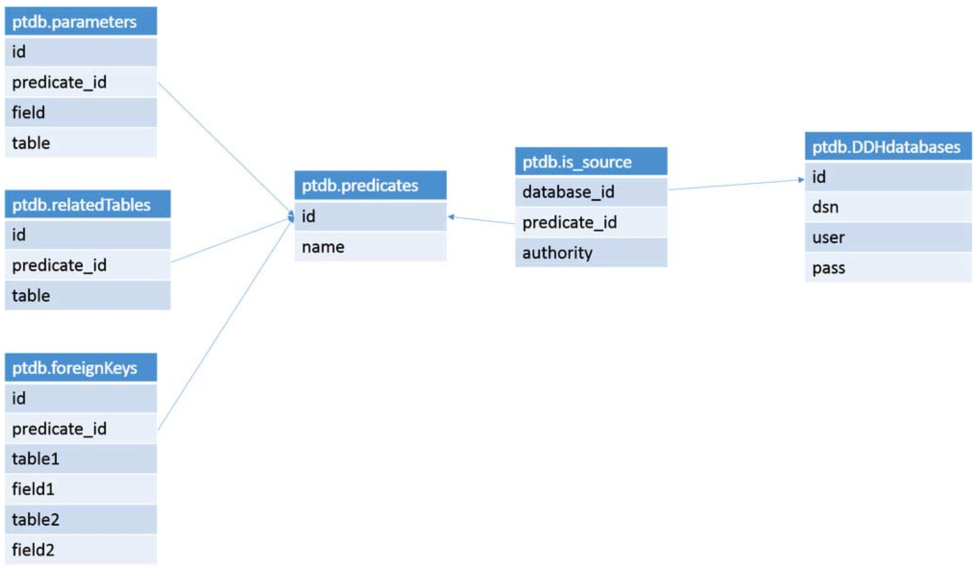


Fig. 2. The predicate translation database schema.

provided by the *Predicate Translation Database* (PTD). Thus, this database is used to implement Definition 9 into the framework; that is, PTD maintains information about relations between predicates and data sources which enable the information retrieval from that sources that the user indicate as potential support sources. In addition the PTD can store additional information regarding some characteristics (*e.g.*, quality or credibility) associated with the database in the matters of the particular predicate, for every predicate that can be supported with information in a database in the DDH. For example, we want to store that for the predicate $weather(City, Weather)$ the database D_1 has an accuracy of 0.75, whereas D_3 is also a possible source for the predicate *weather* but only has a 0.3 accuracy. Consequently, in this version of the framework the Predicate Translation Database is defined by the tables shown in Fig. 2:

– *Predicates table*: it has information about predicate's functors in the program that are able to be supported by some databases in the DDH.
– *DDH databases table*: stores the information needed to connect with every database available in the DDH.
– *Is_source database*: indicates when certain databases can be used to support a given predicate; it also stores information to be used by arguments' preference criteria, *i.e.*, the authority given to the database to support the predicate.
– *Parameters table*: it maintains the equivalence between a predicate's parameter and a pair (table, field).
– *RelatedTables table*: it keeps information about the tables that take part in the SQL JOINS needed to obtain information about a particular predicate.
– *ForeignKeys table*: it maintains a list of the pairs (primaryKey, foreignKey) on which the SQL JOINS have to be made.

With the introduced architecture it is possible to carry out argumentation processes capable of relying on information from relational databases to provide argument support. The interaction among the components in the framework when solving a query is depicted in Fig. 3. The process is executed each time the eDBI-DeLP Server receives a query.

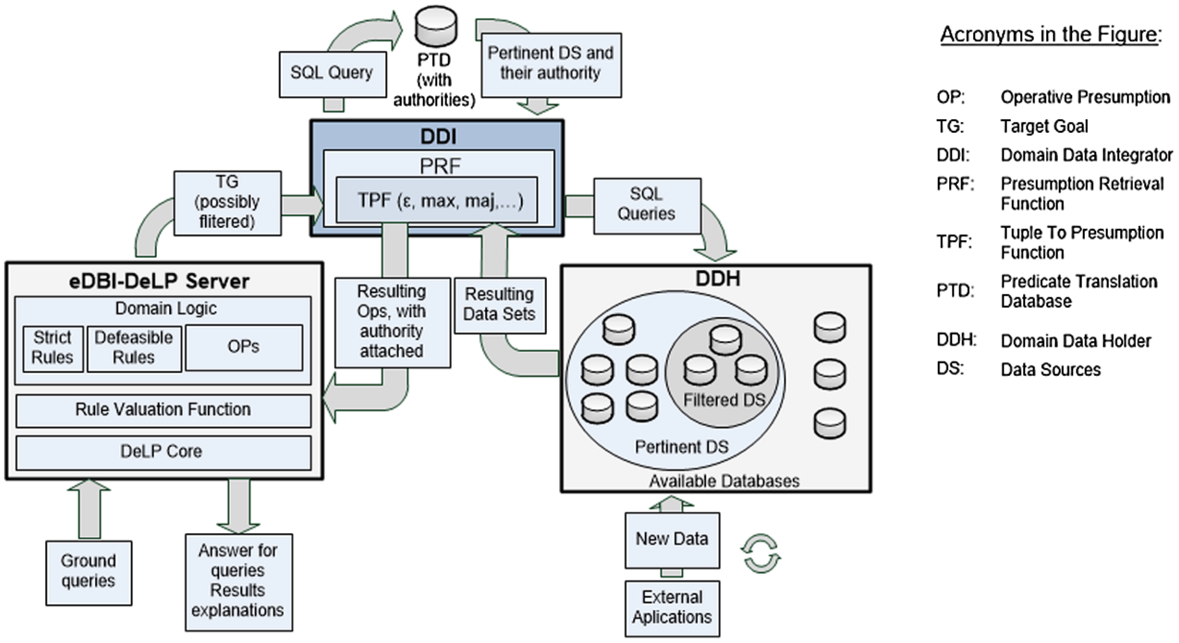


Fig. 3. The eDBI-DeLP argumentation process.

Remark 1. The process of answering a query is considered as part of a closed transaction in the sense of database theory. This is because the available databases (*i.e.*, their current state) are part of the *edbi-delp*. Thus, if we change some database then we are modifying the program as well, but since queries are posed to particular programs then they are solved with the information available when the query arrives.

To implement the *Presumption Retrieval Function* formalized before, we define a high-level algorithm for their specification. Operationally speaking, to achieve their goals, the *PRF* function executes SQL queries to the PTD and databases in the DDH and adapt the resulting datasets so the DeLP Core can use them. In the process involved they use three other functions:

- The *Obtain Instantiated Parameters function* receives a list of parameters from a function objective and returns those that are ground. For example, for the predicate director(Movie, tarantino) the function returns the list [tarantino], while for the predicate film_genre(pirates_of_the_caribbean, comedy) it returns [pirates_of_the_caribbean, comedy].
- The *Obtain Instantiated Fields function* takes a list of fields and a list of parameters and returns those fields corresponding to instantiated parameters. For example, if the function receives [[table.field1, table.field2], [Movie, tarantino]] then it returns [table.field2].
- The *Generate Operative Presumptions function* receives a functor's name, a list of values and an authority $\mathcal{V}$, and returns a pair $(AL, \mathcal{V})$ where functor's name is the predicate's functor for AL and the list of values is the predicate's parameters. For example, if it receives [film_genre, ['Game of Thrones', 'Drama'], 0.8] then (film_genre('Game of Thrones', 'Drama'), 0.8) is the generated Operative Presumption.

After outlining the auxiliary functions for the algorithm, we show in Algorithm 1 the implementation for the process used by eDBI-DeLP to obtain Operative Presumptions for a *TG*, *i.e.*, the Presumption

Retrieval function. We also present the implementation of the filters previously introduced in the paper, which will help Algorithm 1 to achieve its goal.

Algorithm 1 Presumption retrieval function

```
1: function PresumptionRetrievalFunction(TargetGoal $L_i$):OPList
2:     Decompose $L_i$ into its Functor func, a list of parameters $p_1, \dots, p_n$ and its filter filt if $L_i$ is an annotated literal, or filt = null otherwise
3:     instantiatedParameters ← obtainInstantiatedParameters($p_1, \dots, p_n$)
4:     SourcesList ← Execute a SQL Query in the form "SELECT DDHdatabases.id, DDHdatabases.DSN, DDHdatabases.User, DDHdatabases.Pass, is_source.Authority FROM predicates JOIN is_source ON predicates.id = is_source.predicate_id JOIN DDHdatabases ON DDHdatabases.id = is_source.database_id WHERE name = func"
5:     //We begin by retrieving every tuple listed as a possible support for the Target Goal that matches with the instantiated parameters of the former
6:     for each DSN ∈ SourcesList do
7:         dsn_id ← present DSN Id
8:         fieldsToRetrieve ← Execute a SQL Query in the form "SELECT Table, Field FROM parameters WHERE predicate_id = dsn_id"
9:         whereFields ← obtainInstantiatedFields(fieldsToRetrieve, $p_1, \dots, p_n$)
10:        joinTables ← Execute a SQL Query in the form "SELECT Table FROM relatedTables WHERE predicate_id = dsn_id"
11:        joiningFields ← Execute a SQL Query in the form "SELECT Tables, Fields FROM foreignKeys WHERE predicate_id = dsn_id"
12:        Connect to database indicated by DSN using user User and password Pass
13:        RetrievedTuple ← Execute a SQL Query in the form "SELECT fieldsToRetrieve FROM joinTables ON joiningFields WHERE whereFields = instantiatedParameters"
14:        TupSet ← TupSet + (RetrievedTuple, dsn_id)
15:    end for
16:    //Once we retrieved any significant tuple we pass them to the proper filter according to $L_i$
17:    switch filt do
18:        case $\epsilon$ or null
19:            OPList ← TuplesToPresumptions(TupSet, $L_i$)
20:        case max
21:            OPList ← MaxFilter(TupSet, $L_i$)
22:        case maj
23:            OPList ← MajFilter(TupSet, $L_i$)
24:        case sources(DBList)
25:            OPList ← SourcesFilter(TupSet, $L_i$)
26:        case excluded_sources(DBList)
27:            OPList ← ExcludedSourcesFilter(TupSet, $L_i$)
28:    return OPList
29: end function
```

Now that we have presented the algorithms used to retrieve argument supporting information from the databases we will briefly look into their computational complexity aspects. In [13] it was shown that the non-filtered version of the PRF has a complexity of $O(p + n^{s+1})$ where p is the number of predicates' parameters, s is the maximum number of tables implied in a join, and n the maximum number of entries in any table on the DDH. We can show that this complexity does not get increased by the inclusion of filtering in the PRF. To see this consider first the complexity of the TPF algorithm. It is easy to see that such algorithm (and also every filtering algorithm presented) depends on the cardinality of *TupSet*, which control the number of iterations in the **for** cycle, *i.e.*, Algorithms 2–6 are in $O(n)$ since n is the number of tuples in *TupSet* in the worst case. And, since the call for the proper filter is not nested in the retrieve of argument supporting tuples, then execution time for filters is added to the time needed to retrieve tuples (instead of multiply it). Thus, since the addition of two complexity orders is in the order of the greater addend and $O(n) < O(p+n^{s+1})$ we can conclude that the overall complexity order of the new PRF algorithm presented in this work (*i.e.*, tuple retrieving plus filtering) is in the same complexity class than the algorithm presented in [13], and then filtering does not increase complexity.

Algorithm 2 Tuples to presumptions function

```
1: function TuplesToPresumptions(TupSet, $L_i$):OPList
2:     Decompose $L_i$ into its Functor func, a list of parameters $p_1, \dots, p_n$ and its filter $\epsilon$ if $L_i$ is an annotated literal, or filt = null otherwise
3:     for each tuple (tup, DB) $\in$ TupSet do
4:         auth $\leftarrow$ Execute a SQL Query in the form “SELECT is_source.Authority FROM is_source WHERE name = func and database_id = DB”
5:         OPList $\leftarrow$ OPList + generateOperativePresumption(func, tup, auth)
6:     end for
7:     return OPList
8: end function
```

Algorithm 3 Max filter function

```
1: function MaxFilter(TupSet, $L_i$):OPList
2:     Decompose $L_i$ into its Functor func, a list of parameters $p_1, \dots, p_n$ and its filter max
3:     for each tuple (tup, DB) $\in$ TupSet do
4:         auth $\leftarrow$ Execute a SQL Query in the form “SELECT is_source.Authority FROM is_source WHERE name = func and database_id = DB”
5:         if max $=$ auth then
6:             OPList $\leftarrow$ OPList + generateOperativePresumption(func, tup, auth)
7:         else if max $<$ auth then
8:             max $\leftarrow$ auth
9:             OPList $\leftarrow$ generateOperativePresumption(func, tup, auth)
10:        end if
11:    end for
12:    return OPList
13: end function
```

Algorithm 4 Maj filter function

```
1: function MajFilter(TupSet, $L_i$):OPList
2:     Decompose $L_i$ into its Functor func, a list of parameters $p_1, \dots, p_n$ and its filter maj
3:     for each tuple (tup, DB) $\in$ TupSet do TupAppearances $=$ amount of times that tup appears in TupSet
4:         if TupAppearances $<$ MaxAppearances then
5:             auth $\leftarrow$ Execute a SQL Query in the form “SELECT is_source.Authority FROM is_source WHERE name = func and database_id = DB”
6:             maxAuth $\leftarrow$ auth
7:             OPList $\leftarrow$ generateOperativePresumption(func, tup, auth)
8:         else if TupAppearances $=$ MaxAppearances then
9:             auth $\leftarrow$ Execute a SQL Query in the form “SELECT is_source.Authority FROM is_source WHERE name = func and database_id = DB”
10:            if maxAuth $<$ auth then
11:                maxAuth $\leftarrow$ auth
12:                OPList $\leftarrow$ OPList + generateOperativePresumption(func, tup, auth)
13:            end if
14:        end if
15:    end for
16:    return OPList
17: end function
```

Algorithm 5 Sources filter function

```
1:  function SOURCESFILTER($TupSet$, $L_i$):$OPList$
2:      Decompose $L_i$ into its Functor func, a list of parameters $p_1, \ldots, p_n$ and its filter $source(DBList)$
3:      for each tuple $(tup, DB) \in TupSet$ do
4:          if $DB \in DBList$ then
5:              $auth \leftarrow$ Execute a SQL Query in the form "SELECT is_source.Authority FROM is_source WHERE name = func and database_id = DB"
6:              $OPList \leftarrow OPList$ + generateOperativePresumption($func$, $tup$, $auth$)
7:          end if
8:      end for
9:      return $OPList$
10: end function
```

Algorithm 6 Excluded sources filter function

```
1:  function EXCLUDEDSOURCESFILTER($TupSet$, $L_i$):$OPList$
2:      Decompose $L_i$ into its Functor func, a list of parameters $p_1, \ldots, p_n$ and its $filterexcluded\_source(DBList)$
3:      for each tuple $(tup, DB) \in TupSet$ do
4:          if $DB \notin DBList$ then
5:              $auth \leftarrow$ Execute a SQL Query in the form "SELECT is_source.Authority FROM is_source WHERE name = func and database_id = DB"
6:              $OPList \leftarrow OPList$ + generateOperativePresumption($func$, $tup$, $auth$)
7:          end if
8:      end for
9:      return $OPList$
10: end function
```

5. Conclusions and related work

We have shown an approach to combine defeasible argumentation with relational database technologies, where the latter is used to obtain information that enables the construction of arguments to support claims. In particular, in this work we focused on providing more control on how such information is obtained by using methods that state *how* the developer expects that certain literal in a rule is to be supported. For providing such control we presented a filtering mechanism. Rather than being tailored for particular scenarios, this fine-grained control is versatile enough to be useful in different application domains. Moreover, even when we have presented a particular set of filters, to expand such set only involves altering Definition 16 providing the semantics for the new filters (and perhaps modifications of the PTD, if new information regarding the sources is required for the filters). This makes the framework flexible enough to handle dynamic environments where new requirements can arise: such requirements may involve new needs that may be better captured by modifying the argument support mechanism adding new filters rather than modifying the rules in the program to capture said needs.

We were also concerned about how the characteristics of the databases involved in the argument construction process influence the dialectical analysis performed by the argumentation system. Since different sources can have a different level of expertise in the different aspects involved in the argumentation, instead of using a preference order between the different databases, we claim that it is also important to consider which literal the process is trying to support; for instance, we can have a greater confidence in a database storing tax income information when trying to support a literal regarding the salary of certain person rather than using a database with information regarding the films that person likes.

To model the topic expertise we have chosen to associate a pair (*predicate*, *database*) with a value representing the authority assigned to the database for the particular topic the predicate is about. Then, when trying to establish the defeat relation between arguments we combine this authority with the value associated with the rules in each argument. This provides a valuation for these arguments which can be used to compare them to adjudicate the defeat of one over the other. This combination is used in a cautious approach where we combine the valuation of the rule that infers the claim of the argument with the least valued element in the body of such rule; nevertheless, even when we presented a particular valuation function, to modify such behavior only involves the modification of Definition 18 to accommodate it to the particular needs of the application domain. For instance, to be less cautious, we can consider the average between the values in the elements of the body of the rule instead of the minimal value. Moreover, if we prefer to follow a credulous approach, we can consider the maximal value in elements instead. As it was explained, to include such modifications in the framework is straightforward, and will not affect how the rest of the framework works, only affecting final answers to queries.

Recently, the research community has developed an interest in integrating relational database technologies with defeasible argumentation systems. In particular, although this paper builds on [13], here we have extended that work in several novel aspects. The main focus of [13] was to formally support the construction of arguments using information coming from relational databases; however, in that paper no further control on how to interact with such databases was provided. Here, we extended and adapted the method to retrieve information from the databases to introduce the notion of filters. In contrast with [13], this extension provides a tool to modify how tuples from databases should be retrieved, and along with filters we also introduced the use of authority.

In [13], no particular comparison criterion was introduced, and the use of databases did not influence the dialectical process itself, even when they did influence the argument building process. Here, we acknowledge such influence providing means to control how the use of particular sources of information results on different answers to queries. We also provide a definition of argument valuation that accounts for the particular characteristics of the databases that support the argument along with a comparison criterion tailored for such valuation. Finally, we have extended and revised other aspects of the framework presented in [13] such as the derivation mechanism, the Predicate Translation Database, and the algorithm used to retrieve Operative Presumptions (among others) to achieve the above mentioned goals.

In [36], Wardeh et al. introduced a protocol called PADUA to support two agents debating a classification by offering arguments based on association rules mined from individual datasets. In contrast with our approach, this research connects databases with argumentation from a machine learning perspective, including as well a large scale evaluation designed to test the effectiveness of using PADUA to detect misclassified examples, and to provide a comparison with other classification systems. In [3] the problem of using defeasible reasoning in a massive data repository was addressed, but instead of using databases to look for information to support conclusions its repository is the Web, more specifically the Semantic Web. Recently, several large-scale domain-dependent datasets have been released, providing additional motivation for the development of the framework proposed in that paper. Thus, using databases we probably will have access to the same repositories that those accessed by the system in [3], and even more.

Regarding databases, there is research related to the use of non-monotonic reasoning to resolve inconsistencies in databases by means of *database repairs* [32]. Although this approach can resolve inconsistencies in databases allowing reasoning on data stored in them, database repair is conducted by the explicit addition, modification or suppression of tuples in databases. Instead, our approach is based on a conflict resolution strategy using argumentation, avoiding thus any modification of the actual information stored in the database. Other approaches to deal with the issue of inconsistent databases are presented in [5,11], where instead of repairing the databases, modifications to queries are made to provide consistent answers to them. In eDBI-DeLP we always obtain consistent answers as inconsistency will be managed automatically by the DeLP Core and the proof procedure using the domain information. In addition, in eDBI-DeLP the filtering mechanism provides tools to discard potentially contradictory information, as we have shown in Example 1 where the max filter is used to retrieve only one temperature value of a given city (the one in the database with the maximum authority).

Another approach that uses databases as the basis of a reasoning process is presented in [12]. The aim of the reasoning mechanism is to address the different conflicts that may arise when merging several databases. Nevertheless, the databases used in that work are deductive databases, *i.e.*, databases that are made of an extensional part – a set of positive or negative ground literals – and an intensional part, *i.e.*, a set of first order function-free clauses. Deductive databases have some known drawbacks that are not present in eDBI-DeLP, such as the need to define criteria for using a rule included in the database as a deduction rule or a coherence rule; in eDBI-DeLP this does not happen because rules are only used to build arguments. Another well-known drawback of deductive databases is that they have the possibility of falling in infinite loops in the deduction process; this is also avoided in eDBI-DeLP using constraints over the argumentation lines [18].

Another work that is related to ours is the one in [8]. In that work the authors introduce a framework to allow for reasoning on heterogeneous sources by combining arbitrary monotonic and non-monotonic logics. To do this the authors exploit bridge rules to infer knowledge on the different contexts in play (*i.e.*, the different logics and the belief bases obtained from them), where the bridge rules in one context are used to augment belief bases in other contexts. Based on this last feature the authors present how to achieve a certain equilibrium among the contexts, which represent acceptable belief states a system may adopt. Thus, our work can be modelled as a particular instance of Brewka and Eiter's framework, as there are certain parallels even when the aims of the works are clearly different. It can be argued that in our work we have two different contexts in play (a non-monotonic one – DeLP – and a monotonic one – SQL), and some "one-way" bridge rules can be obtained that augment the belief base in one context (DeLP) with information obtained from the other context (the databases). In our work such bridge rules are represented through the use of the Pertinence Relation and the Presumption Retrieval Function. Nevertheless, note that there will be some differences between a "pure" multi-context system and our approach. For instance, to properly adequate our work to Brewka and Eiter's we will have to retrieve all tuples related to each annotated literal in the program and build every operative presumption at first (*i.e.*, we need to apply all applicable bridge rules), and then use those necessary to the dialectical process. Instead, for the sake of efficiency in our work we have chosen to only retrieve tuples and build Operative Presumptions when needed by such dialectical proof procedure.

As for future work, there are several lines of research that we are following. First, as we have discussed, the set of filters presented in this work can be easily expanded to accommodate the system to new requirements. As a future line of work we plan to tackle different application domains and develop new filters inspired in the detected needs. Moreover, in the present work we can have certain combination between filters (*v.g.*, by using them sequentially for the same predicate in a rule); nevertheless, we want to further increase the expressiveness in our framework by providing ways to use more refined combinations (*e.g.*, in the same annotated literal) that adhere to the original semantics of the filters combined as much as possible.

Also, several interesting questions and lines of work arise from our use of the notion of *authority*. For instance, in this work we have followed a cautious approach when considering how authority in operative presumptions affects the valuation of arguments, but it may be interesting to explore how less cautious approaches behave. Also, we acknowledge that in any dynamic environment a concept such as authority cannot be static. In this first approach we were not concerned with how authority may change, but we plan to integrate in the framework automatic mechanisms to update it, inspired by similar efforts in areas like Belief Revision and Trust in multiagent systems, *e.g.*, [16,24,31,33,34,37].

It is interesting to study how to enhance this framework with semantic information about predicates allowing to automatically obtain information about the different possible data sources; *e.g.*, one way this could be done is by using ontologies with semantic definitions for every parameter in a predicate. In this way, such ontologies may help us to identify and recognize the structure of the different data sources, allowing the definition of processes that could automatically fill the PTD. Additionally, this can help to add new capabilities, *e.g.*, data alignment among heterogeneous databases. Notice that the presented structure of the PTD is adequate to maintain the information relating predicates and data sources provided by such processes, making the addition and modification of data sources easier. Also, the proposed framework is flexible enough to allow the automatic generation of the necessary SQL queries; thus, every modification in the PTD is reflected in the formed queries directly, because they are constructed on the fly.

We also plan to tackle on the implementation of the framework introduced in this work. As expected, for such implementation we will take as basis the one presented in [13]. Once we have implemented the new features of eDBI-DeLP we plan to perform empirical evaluations to asses the impact of such new features in the overall efficiency of the framework. As explained before, the use of filtering does not increase complexity for the PRF, but clearly execution times could be different than if no filtering is made at all. It is our intuition, however, that the impact of filtering in execution times for the PRF should not be noticeable, and that the most time-consuming task will still be the dialectical process, with little negative impact of the filtering process. Moreover, we believe that in certain situations filters will improve performance, *e.g.*, when we use the *max* filter to only retrieve the most credible operative presumption, thus greatly reducing the number of arguments that need to be evaluated. That is, we believe that if filters are applied correctly we can reduce the amount of arguments under consideration when building dialectical trees, which will indeed reduce the overall time needed to solve queries in the system.

Finally, another line of research we are currently following is the dynamics in the knowledge in our scenario. This involves at least two different aspects. On the one hand, we want to explore how changes in the set of available databases affect the query solving process in our framework. In our current version of the paper the set of available databases is included in the *edbi-delp*, and thus when posing a query to a program the set of databases is fixed for that query. Indeed, if the set of databases changes then the program change, but we still should consider the previous state of the databases for queries already being solved, since they were posed to the previous state of the program. For the future we plan to allow that the databases change in a program while considering these changes for queries being solved. This poses some interesting challenges, both regarding how we treat arguments in dialectical trees supported by information no longer available, and how it affects the credibility and trust assigned to the databases (*i.e.*, how authority reflects changes, changing in turn the defeat relation and thus the query resolution).

On the other hand, we can tackle the update of the rules in the program based on the analysis of the information stored in the DDH. In particular, as supporting information is searched in the databases, counterexamples to already known rules may arise. We plan to take advantage of such counterexamples to further refine the available knowledge. While searching for support for some *TG* the set of rules may be revised when data with values different to those expected are found; for example, strict rules may be weakened to defeasible ones, or new refined defeasible rules may be formed by analyzing the characteristics of the newly found data [15]. To do this, we can exploit mechanisms already developed for DeLP to make the knowledge base updates; for instance, we can take advantage of the addition and removal of elements of knowledge from a DeLP program provided by contextual queries proposed in [20].

Acknowledgements

This work has been partially supported by EU H2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 690974 for the project MIREL: MIning and REasoning with Legal texts, the project "Representación de conocimiento y razonamiento argumentativo: Herramientas inteligentes para la web y las bases de conocimiento", PGI 24/N040 funded by the Universidad Nacional del Sur, and PID UNER 7042 funded by the Universidad Nacional de Entre Ríos (UNER). This work is also funded by the Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), from Argentina.

References

[1] T. Alsinet, C.I. Chesñevar, L. Godo and G.R. Simari, A logic programming framework for possibilistic argumentation: Formalization and logical properties, *Fuzzy Sets and Systems* **159**(10) (2008), 1208–1228. doi:10.1016/j.fss.2007.12.013.
[2] L. Amgoud and S. Kaci, An argumentation framework for merging conflicting knowledge bases: The prioritized case, in: *Proc. of the Ecsqaru-2005 Conf., Lnai 3571*, Springer, 2005, pp. 527–538.
[3] N. Bassiliades, G. Antoniou and I.P. Vlahavas, A defeasible logic reasoner for the semantic web, *Int. J. Semantic Web Inf. Syst.* **2**(1) (2006), 1–41. doi:10.4018/jswis.2006010101.
[4] T.J.M. Bench-Capon and P.E. Dunne, Argumentation in artificial intelligence, *Artif. Intell.* **171** (2007), 619–641. doi:10.1016/j.artint.2007.05.001.
[5] L.E. Bertossi, Consistent query answering in databases, *Sigmod Record* **35**(2) (2006), 68–76. doi:10.1145/1147376.1147391.
[6] F. Bex, I. Bratko, C.I. Chesñevar, W. Dvorak, M.A. Falappa, S.A. Gaggl, A.J. García, M.P. González, T. Gordon, J. Leite, M. Možina, C. Reed, G.R. Simari, S. Szeider, P. Torroni and S. Woltran, Agreeement technology handbook, S. Modgil and F. Toni, eds, Springer Verlag, Law, Governance and Technology Series, 2012, Chap. Argumentation in Agreement Technologies.
[7] F. Bex, J. Lawrence, M. Snaith and C. Reed, Implementing the argument web, *Communications of the Acm* **56**(10) (2013), 66–73. doi:10.1145/2500891.
[8] G. Brewka and T. Eiter, Equilibria in heterogeneous nonmonotonic multi-context systems, in: *AAAI*, Vol. 7, 2007, pp. 385–390.
[9] C.E. Briguez, M.C. Budán, C.A.D. Deagustini, A.G. Maguitman, M. Capobianco and G.R. Simari, Argument-based mixed recommenders and their application to movie suggestion, *Expert Syst. Appl.* **41**(14) (2014), 6467–6482. doi:10.1016/j.eswa.2014.03.046.
[10] M. Capobianco, C.I. Chesñevar and G.R. Simari, Argumentation and the dynamics of warranted beliefs in changing environments, *Autonomous Agents and Multi-Agent Systems* **11**(2) (2005), 127–151. doi:10.1007/s10458-005-1354-8.
[11] A. Celle and L.E. Bertossi, Querying inconsistent databases: Algorithms and implementation, in: *Computational Logic*, 2000, pp. 942–956.
[12] L. Cholvy and C. Garion, Answering queries addressed to several databases according to a majority merging approach, *J. Intell. Inf. Syst.* **22** (2004), 175–201. doi:10.1023/B:JIIS.0000012469.11873.bb.
[13] C.A.D. Deagustini, S.E.F. Dalibón, S. Gottifredi, M.A. Falappa, C.I. Chesñevar and G.R. Simari, Relational databases as a massive information source for defeasible argumentation, *Knowl.-Based Syst.* **51** (2013), 93–109. doi:10.1016/j.knosys.2013.07.010.

[14] P.M. Dung, R.A. Kowalski and F. Toni, Dialectic proof procedures for assumption-based, admissible argumentation, *Artificial Intelligence* **170**(2) (2006), 114–159. doi:10.1016/j.artint.2005.07.002.

[15] M.A. Falappa, G. Kern-Isberner and G.R. Simari, Explanations, belief revision and defeasible reasoning, *Artif. Intell.* **141**(1/2) (2002), 1–28. doi:10.1016/S0004-3702(02)00258-8.

[16] R. Falcone and C. Castelfranchi, Trust dynamics: How trust is influenced by direct experiences and by trust itself, in: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, IEEE Computer Society, 2004, pp. 740–747.

[17] E. Ferretti, M.L. Errecalde, A.J. García and G.R. Simari, Decision rules and arguments in defeasible decision making, in: *Proceedings of the 2008 Conference on Computational Models of Argument: Proceedings of Comma 2008*, IOS Press, 2008, pp. 171–182.

[18] A.J. García and G.R. Simari, Defeasible logic programming an argumentative approach, *Tplp* (2004), 95–138.

[19] A.J. García and G.R. Simari, Defeasible logic programming: Delp-servers, contextual queries, and explanations for answers, *Argument & Computation* **5**(1) (2014), 63–88. doi:10.1080/19462166.2013.869767.

[20] S. Gottifredi, A.J. García and G.R. Simari, Query-based argumentation in agent programming, in: *12th Ibero-American Conference on Ai (Iberamia 2010)*, 2010, pp. 284–295.

[21] N.K. Janjua, F.K. Hussain and O.K. Hussain, Semantic information and knowledge integration through argumentative reasoning to support intelligent decision making, *Information Systems Frontiers* **15**(2) (2013), 167–192. doi:10.1007/s10796-012-9365-x.

[22] J. Lawrence, F. Bex, C. Reed and M. Snaith, Aifdb: Infrastructure for the argument web, in: *Comma*, 2012, pp. 515–516.

[23] V. Lifschitz, Foundations of logic programming, *Principles of Knowledge Representation* **3** (1996), 69–127.

[24] J. Ma and M.A. Orgun, Trust management and trust theory revision, *Trans. Sys. Man Cyber. Part A* **36**(3) (2006), 451–460. doi:10.1109/TSMCA.2006.871628.

[25] M.V. Martinez, A.J. García and G.R. Simari, On the use of presumptions in structured defeasible reasoning, in: *Computational Models of Argument – Proceedings of COMMA 2012*, Vienna, Austria, September 10–12, 2012, B. Verheij, S. Szeider and S. Woltran, eds, Frontiers in Artificial Intelligence and Applications, Vol. 245, IOS Press, 2012, pp. 185–196.

[26] S. Modgil, F. Toni, F. Bex, I. Bratko, C.I. Chesñevar, W. Dvořák, M.A. Falappa, X. Fan, S.A. Gaggl, A.J. García, M.P. González, T.F. Gordon, J.A. Leite, M. Možina, C. Reed, G.R. Simari, S. Szeider, P. Torroni and S. Woltran, The added value of argumentation: Examples and challenges in agreement technologies, S. Ossowski ed., in: *Law, Governance and Technology*, Vol. 8, Springer, New York, 2013, pp. 357–403, Chap. 21. ISBN 978-94-007-5583-3 (eBook).

[27] C.I. Chesñevar, A. Maguitman and M. González, Argumentation in artificial intelligence, Springer Verlag, 2009, pp. 403–422, Chap. Empowering Recommendation Technologies Through Argumentation.

[28] H. Prakken and G. Sartor, Argument-based extended logic programming with defeasible priorities, *Journal of Applied Non-Classical Logics* **7**(1) (1997).

[29] I. Rahwan, Mass argumentation and the semantic web, *J. Web Sem.* **6**(1) (2008), 29–37. doi:10.1016/j.websem.2007.11.007.

[30] I. Rahwan and G.R. Simari, *Argumentation in Artificial Intelligence*, Springer, 2009. doi:10.1007/978-0-387-98197-0.

[31] J. Sabater and C. Sierra, Review on computational trust and reputation models, *Artificial Intelligence Review* **24**(1) (2005), 33–60. doi:10.1007/s10462-004-0041-5.

[32] E. Santos, J.P. Martins and H. Galhardas, An argumentation-based approach to database repair, in: *19th European Conference on Artificial Intelligence (ECAI)*, 2010, pp. 125–130.

[33] L.H. Tamargo, A.J. García, M.A. Falappa and G.R. Simari, On the revision of informant credibility orders, *Artif. Intell.* **212** (2014), 36–58. doi:10.1016/j.artint.2014.03.006.

[34] L.H. Tamargo, S. Gottifredi, A.J. García and G.R. Simari, Sharing beliefs among agents with different degrees of credibility, *Knowl. Inf. Syst.* **50**(3) (2017), 999–1031. doi:10.1007/s10115-016-0964-6.

[35] M. Thimm, Realizing argumentation in multi-agent systems using defeasible logic programming, in: *Argmas*, P. McBurney, I. Rahwan, S. Parsons and N. Maudet, eds, Lecture Notes in Computer Science, Vol. 6057, Springer, 2009, pp. 175–194. ISBN 978-3-642-12804-2.

[36] M. Wardeh, T.J.M. Bench-Capon and F. Coenen, Padua: A protocol for argumentation dialogue using association rules, *Artif. Intell. Law* **17**(3) (2009), 183–215. doi:10.1007/s10506-009-9078-8.

[37] F.M. Zahedi and J. Song, Dynamics of trust revision: Using health infomediaries, *Journal of Management Information Systems* **24**(4) (2008), 225–248. doi:10.2753/MIS0742-1222240409.