Knowledge-Based Systems 51 (2013) 93–109

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Relational databases as a massive information source for defeasible argumentation

Cristhian A.D. Deagustini [a,b,c,*], Santiago E. Fulladoza Dalibón [a,b,c], Sebastián Gottifredi [a,c], Marcelo A. Falappa [a,c], Carlos I. Chesñevar [a,c], Guillermo R. Simari [a]

[a] Artificial Intelligence Research and Development Laboratory, Department of Computer Science and Engineering, Universidad Nacional del Sur, Alem 1253, 8000 Bahía Blanca, Buenos Aires, Argentina
[b] Agents and Intelligent Systems Area, Faculty of Management Sciences, Universidad Nacional de Entre Ríos, Tavella 1424, 3200 Concordia, Entre Ríos, Argentina
[c] Consejo Nacional de Investigaciones Científicas y Técnicas, Av. Rivadavia 1917, (C1033AAJ) Ciudad Autónoma de Buenos Aires, Argentina

ARTICLE INFO

Article history:
Received 5 February 2013
Received in revised form 15 July 2013
Accepted 16 July 2013
Available online 26 July 2013

Keywords:
Knowledge-based systems
Defeasible argumentation
Relational databases
Massive argumentation
Argument-supporting data retrieval

ABSTRACT

Argumentation provides a sophisticated yet powerful mechanism for the formalization of commonsense reasoning in knowledge-based systems, with application in many areas of Artificial Intelligence. Nowadays, most argumentation systems build their arguments on the basis of a single, fixed knowledge base, often under the form of a logic program as in Defeasible Logic Programming or in Assumption-Based Argumentation. Currently, adding new information to such programs requires a manual encoding, which is not feasible for many real-world environments which involve large amounts of data, usually conceptualized as relational databases.

This paper presents a novel approach to compute arguments from premises obtained from relational databases, identifying several relevant aspects. In our setting, different databases can be updated by external, independent applications, leading to changes in the spectrum of available arguments. We present algorithms for integrating a database management system with an argument-based inference engine. Empirical results and running-time analysis associated with our approach show that it provides a powerful alternative for efficiently achieving massive argumentation, taking advantage of modern DBMS technologies. We contend that our proposal is significant for developing new architectures for knowledge-based applications, such as Decision Support Systems and Recommender Systems, using argumentation as the underlying inference model.

1. Introduction

Argumentation represents a sophisticated device for the formalization of commonsense reasoning, providing a practical, human-like reasoning mechanism with the ability of suitably handling incomplete and/or potentially inconsistent information. As such, it has found application, and proven its value, in different areas of Artificial Intelligence (AI) such as multi-agent systems, recommender systems, and decision support systems among others (see e.g., [1–4]).

Intuitively, an argument is a coherent set of statements through which a claim is made, and support is offered for it, trying to influence an audience in a context of disagreement. Thus, the ultimate acceptance of an argument will depend on a dialectical analysis considering the arguments in favor and against the claim [2,5].

Since the late 1980s, several frameworks (generically called Argumentation Systems) have been developed for formalizing argumentative reasoning; they provide different knowledge representation and inference capabilities (e.g., see [6–14]). This research activity was complemented with an increased interest in developing different implementations, having most of them a natural connection with logic programming (see [15–18]), where rules and facts are used to represent knowledge, and logical inference provides a mechanism for inferring conclusions. This type of systems, where the structure of arguments is built from a specific knowledge base, is referred to as Rule Based Argumentation System, or RBAS (see Part II in [2]).

Clearly, RBAS are useful systems, but there are challenges for their effective exploitation to solve real world problems. A major difficulty is that inferences are performed from a local knowledge base whose updating process is not straightforward [19] because it is necessary to explicitly encode the new information in the program. Therefore, providing tools for automatically updating the knowledge base with information coming from external applications will add an interesting capability.

* Corresponding author at: Artificial Intelligence Research and Development Laboratory, Department of Computer Science and Engineering, Universidad Nacional del Sur, Alem 1253, 8000 Bahía Blanca, Buenos Aires, Argentina. Tel.: +54 93456250503.
E-mail address: caddeagustini@gmail.com (C.A.D. Deagustini).

0950-7051/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2013.07.010

A possible solution for such challenges comes from considering a framework that integrates a RBAS with database technologies; this approach will improve the capabilities offered by traditional argumentation systems by connecting them with repositories stored in relational databases. While for our formalization we will use as a basis a particular RBAS called Defeasible Logic Programming (DeLP) [11], this will not represent any loss of generality. The resulting framework, called DBI-DeLP (Database Integration for DeLP), integrates a database management system (DBMS) and an argument-based inference engine. The DBMS will provide the facts that the RBAS will take as premises for constructing the arguments over which it will reason. In such scenario, it is possible to formalize processes that will feed the argumentative inference engine with supporting data coming from different databases, i.e., we could use the information stored in the databases together with the rules in a DeLP program (the RBAS in question). Thus, arguments will be created exploiting the interaction between rules and records in databases.

We will present a formalization and an implementation for a process capable of benefiting from the interaction just described. The proposal is based on a preliminary approach presented in [20], where a theoretical model for DBI-DeLP was first introduced. Here, we will extend and refine the concepts and definitions needed to further flesh-out the DBI-DeLP framework. Furthermore, we will also include an empirical analysis of the resulting framework, considering different complexity issues associated with solving queries with DBI-DeLP programs, particularly those of the DeLP core proof procedure.

We have organized the paper as follows: Section 2 succinctly introduces DeLP, the formalism behind DBI-DeLP; Section 3 outlines a possible structure to exploit information stored in databases by the argumentation process; Section 4 presents an analysis on running-time efficiency and complexity for DBI-DeLP; and Section 5 contains our main conclusions identifying also lines for future research.

2. Background

In this section we give a brief summary of DeLP, which is a formalism that combines Logic Programming and Defeasible Argumentation providing the declarative representation of information as rules and a defeasible argumentation inference mechanism for warranting the entailed conclusions. These rules are the key element for introducing defeasibility and are used to represent a relation between pieces of knowledge that could be defeated after all things are considered. Using these rules, reasoning is defeasible in a way that is not explicitly programmed. The reader is referred to [11] where a complete presentation of DeLP can be found.

In DeLP, knowledge is represented using facts, strict rules, and defeasible rules. Facts are ground literals representing atomic information or the negation of atomic information using strong negation "$\sim$" (e.g., a, or $\sim a$). Strict rules are denoted $Head \leftarrow Body$, and represent a relation such that if $Body$ can be obtained then $Head$ should be accepted. Defeasible Rules are denoted $Head \prec Body$ and represent a tentative relation that may be used when nothing could be posed against it. Thus, a defeasible rule (or d-rule) expresses that *reasons to accept the d-rule's antecedent Body give reasons to accept its consequent Head*. Finally, we introduce defeasible rules with empty bodies that are called presumptions. Then, given a literal L, a presumption is denoted "$L\prec$" and interpreted as *if nothing is against L there are defeasible reasons to believe in it*; so, presumptions are assumed to be true if nothing could be posed against them. Recent research on the topic of presumptions can be found in [21].

Definition 1 (*Facts, Presumptions, Strict Rules and Defeasible Rules*). A literal L is a ground atom A or a negated ground atom $\sim A$, where "$\sim$" represents the strong negation. Given a literal L_0 and a finite non empty set of literals $L_1, \ldots, L_n$:

- A fact is a literal L_0 denoted "$L\leftarrow$", or "L_0"
- A strict rule is denoted "$L \leftarrow L_1, \ldots, L_n$"
- A defeasible rule is denoted "$L_0 \prec L_1, \ldots, L_n$"
- A presumption is a defeasible rule with empty body, denoted "$L_0\prec$"

with $n > 0$.

Given a literal L, $\sim L$ represents the complement w.r.t. strong negation, and the pair $(L, \sim L)$ is said to be contradictory. Both strict and defeasible rules are ground. Nevertheless, following the usual practice in Logic Programming [17], 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,

$$\mathsf{child\_restricted}(\mathsf{Movie}) \leftarrow \mathsf{has\_violence}(\mathsf{Movie})$$

is a strict rule that represents that *any movie that has violence is a child restricted movie*; and

$$\mathsf{good\_movie}(\mathsf{Movie}) \prec \mathsf{genre}(\mathsf{Movie}, \mathsf{action}), \quad \mathsf{performs\_in}(\mathsf{Movie}, \mathsf{arnold})$$

represents a defeasible rule that can be interpreted as *if there are no reasons to believe otherwise, any action movie in which Arnold Schwarzenegger stars is a good movie*.

A DeLP program (or *delp*) is a pair (Π, Δ) where Π is a set of strict rules and facts, and Δ is a set of defeasible rules and presumptions. Formally:

Definition 2 (*DeLP program*). A *delp* $\mathcal{P}$ is a pair (Π, Δ) where

1. Π is a non-contradictory set of facts and strict rules,
2. Δ is a set of presumptions and defeasible rules.

Strong negation can appear in facts and presumptions, or generally in the head of strict and defeasible rules; therefore, it is important to note that from a *delp* it is possible to obtain contradictory literals. However, 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 is a methodological restriction to avoid the problem of obtaining the full language when Π is contradictory, i.e., *ex falso quodlibet* (see [22] for a summary); but this restriction does not apply to Δ.

From a *delp* it is possible to infer tentative information. These inferences, called defeasible derivations, are computed by backward chaining applying the usual Selective Linear Definite (SLD) inference procedure used in logic programming.

Definition 3 (*Defeasible derivation*). Let $\mathcal{P} = (\Pi, \Delta)$ be a *delp* and L a ground literal. A defeasible derivation of L from $\mathcal{P}$, denoted $\mathcal{P}|\sim L$, consists of a finite sequence $L_1, L_2, \ldots, L_n = L$ of ground literals, where each literal L_i is in the sequence because:

(a) there exists a fact L_i or a presumption $L_i\prec$, or
(b) there exists a rule R_i in $\mathcal{P}$ (strict or defeasible) with head L_i and body $B_1, B_2, \ldots, B_k$ and every literal of the body is an element L_j of the sequence appearing before L_i ($j <$ i).

Since contradictory literals can be derived, the derivation does not provide a strong enough notion to characterize the final inferences of the system. For this, when contradictory literals are derived, DeLP builds arguments for the tentative conclusion and then a dialectical process is used for deciding which literals are

warranted. We will say that a literal L is warranted if there exists an undefeated argument A for L; this will be denoted $\mathcal{P}\mid\sim_w L$. An argument for a literal L, denoted $\langle A,L\rangle$, is a minimal non-contradictory set of d-rules $A \subseteq \Delta$, that allows to derive L. A sub-argument of an argument $\langle A,L\rangle$ is a subset of the d-rules in A.

Definition 4 (*Argument*). Given a *delp* $\mathcal{P}$, an argument A for a grounded literal L, denoted $\langle A,L\rangle$, is a non empty set of defeasible rules in Δ such that:

1. There exists a defeasible derivation for L from $\Pi \cup A$, i.e., $\Pi \cup A \mid\sim L$,
2. $\Pi \cup A$ is non-contradictory,
3. A is minimal with respect to set inclusion in satisfying (1) and (2).

A sub-argument $\langle B,Q\rangle$ of $\langle A,L\rangle$ is an argument for Q such that $B \subseteq A$.

To establish if $\langle A,L\rangle$ is a non-defeated argument, counterarguments against $\langle A,L\rangle$ are considered. A counterargument $\langle C,R\rangle$ of $\langle A,L\rangle$ is an argument that disagrees with a sub-argument $\langle B,Q\rangle$ of $\langle A,L\rangle$, i.e., their conclusions R and Q are contradictory. A defeater for an argument $\langle A,L\rangle$ is a counterargument $\langle C,R\rangle$ that at least is as preferable as (or is unrelated to) $\langle A,L\rangle$ under some comparison criterion.

In DeLP the comparison criterion is a modular part of the argumentation inference engine, and it could be replaced. For instance, we can use a criterion known as *priority among rules*, where an argument defeats another when the former uses rules that are marked as more important than the ones used by the latter. An alternative is to use the *generalized specificity* criterion, where no explicit order among rules or arguments need to be given, and the comparison between two arguments is done syntactically (see [11] for a discussion of the two criteria).

A counterargument can attack the conclusion of an argument or an inner point of it, i.e., it can attack the conclusion of some sub-argument of the argument. Since defeaters are arguments, there may exist defeaters for them, defeaters for those defeaters, and so on. Thus, a sequence of arguments called argumentation line can arise. Clearly, for a particular argument $\langle A,L\rangle$ there might be more than one defeater. Therefore, many argumentation lines could arise from one argument. This leads to a tree structure called dialectical tree (d-tree), denoted $\mathcal{T}_{\langle A,L\rangle}$. In a dialectical tree every non-root node is a defeater of its parent under the comparison criterion chosen to decide defeat; therefore, every path from a leaf node to the root node is a different argumentation line. Once the dialectical tree has been computed, it is necessary to perform a bottom-up analysis of it to decide whether the argument at the root is defeated or not, i.e., the literal that the argument supports is warranted from a *delp* $\mathcal{P}$. Every leaf of the tree is marked *undefeated* and every inner node is marked *defeated* if it has at least one child node marked *undefeated*; otherwise it is marked *undefeated*. Thus, the root node is warranted only if all its attackers are defeated. More details about the warrant procedure can be found in [11].

Example 1. To briefly illustrate how queries are solved in DeLP, consider the *delp* in Fig. 1 which represents information about the stock market domain.

Suppose that we have to answer a query for buy_stock(acme) based on this program. Then, applying the SLD procedure, DeLP will find all rules that have buy_stock (and ∼buy_stock) as its first component, and try to build arguments based on them. So, it will start using the rule

$$\text{buy\_stock(C)} \prec \text{good\_price(C)}$$

to build an argument in favor of buying stocks from "acme", as there is a fact saying that such stocks are at good price. Once that argument is built the dialectical process will try to find counterarguments, either from ∼buy_stock(C) (attack to the conclusion of the previous argument) or from ∼good_price(C) (attack to a sub-argument). In the first case, it will use the rule

$$\sim \text{buy\_stock(C)} \prec \text{good\_price(C)}, \quad \text{risky\_company(C)}$$

to build an argument: good_price(acme) is obtained the same way as before, and the rule

$$\text{risky\_company(C)} \prec \text{in\_fusion(C, AC)}$$

leads to risky_company(acme) using the fact that acme is in fusion with steel, i.e., in_fusion(acme, steel). Then, we have the counterargument

$$\langle\{\sim \text{buy\_stock(acme)} \prec \text{good\_price(acme)}, \quad \text{risky\_company(acme)}, \quad \text{risky\_company(acme)} \prec \text{in\_fusion(acme, steel)}\}, \quad \sim \text{buy\_stock(acme)}\rangle$$

Now, DeLP cannot build a new counterargument attacking the head of this last argument, as there is no other rule with buy_stock in its head; instead, the dialectical process will try to find a counterargument to the sub-argument:

$$\langle\{\text{risky\_company(acme)} \prec \text{in\_fusion(acme, steel)}\}, \quad \text{risky\_company(acme)}\rangle$$

finding the argument for risky_company(acme) shown below which plays that role:

$$\langle\{\sim \text{risky\_company(acme)} \prec \text{in\_fusion(acme, steel)}, \quad \text{strong(steel)}\}, \sim \text{risky\_company(acme)}\rangle$$

as the company that is merging with acme is steel, a company that we know is strong. So, finally the conclusion is that we can buy stocks from acme as they have good price, and the reason we previously had to avoid buying the stock is not valid since we know that acme is not a risky company because it is in fusion with the company steel that is a strong company.

3. Defeasible argumentation over databases

As we have seen in the previous section, DeLP enables query resolution by an argumentative process which deals with incomplete and potentially contradictory information. Several real-world applications, such as recommender and decision-support systems [23], multi-agent systems [24], etc., have been proposed using DeLP as the implementation basis. However, many of these realistic scenarios require massive updates of data; following the previous example, a rule-based movie recommender can improve its recommendations if we feed it with updated information both about the movies that can be recommended and the users for which the recommendations are made.

Consider the stock market example presented in Example 1. Clearly, to assess the status of a company in the stock market, it

```
      buy_stock(C) —≺ good_price(C).
     ∼buy_stock(C) —≺ good_price(C),
                      risky_company(C).
  risky_company(C) —≺ in_fusion(C, AC).
  risky_company(C) —≺ closing(C, AC)
 ∼risky_company(C) —≺ in_fusion(C, AC),
                      strong(AC).
     good_price(acme).
in_fusion(acme, steel).
     strong(steel).
```

Fig. 1. A *delp* $\mathcal{P}$ for the stock market domain.

might be useful to access databases from credit rating agencies (e.g. Moody, Standard & Poor's, The Fitch Group, etc.). Usually, the creditworthiness of a company variates dynamically as the stock market evolves. This variation is also reflected in the associated databases. Thus, DeLP requires additional features to handle such data, since it is not efficient to statically include new data into the program.

Two distinct problems involving the process of building arguments arise in the previous setting. First, we have to establish the connection with the information sources that will be used by this process; second, we need to define an efficient way to feed that information to the process.

Information sources may have different formats (raw text on the World Wide Web, ontologies in the Semantic Web, CSV files, etc.). In fact, recent research has exploited the use of ontologies [25], and the Semantic Web [26], in formalizing argumentation processes. In this context, however, relational databases are by far the predominant choice for storing, organizing and accessing structured data. In fact, several websites, such as the Internet Movie DataBase [27] and MovieLens [28], provide public datasets that are supported by relational databases. To the best of our knowledge there is currently no formal approach to integrate argumentation with the relational database model. This paper presents a framework that integrates Relational Databases technologies with Defeasible Argumentation, called DBI-DeLP Framework. Our approach is based on DeLP, but it could also be extended to other argumentation frameworks (e.g. ABA [13], or ASPIC [14]).

3.1. Enabling argumentation over relational databases

We start with the theoretical foundations of DBI-DeLP. In characterizing the framework, it is necessary to consider the possible presence of contradictory information tied to the use of several databases. Given a database D, it might be the case that tuples $t_1 \in D_1$ and $t_2 \in D_1$ reflect contradictory knowledge, what makes D inconsistent. Another possible scenario is to have two consistent databases D_1 and D_2, such that $t_1 \in D_1$ and $t_2 \in D_2$ are contradictory. In such scenarios, t_1 and t_2 cannot be accepted simultaneously. To handle such data we adopt the notion of *presumption* [11,21] for representing "defeasible" information, avoiding in that manner inconsistencies not allowed in the strict knowledge available, as required by DeLP.

Example 2. Consider the database in Fig. 2 with information about the time and places where employees of a certain company have been. As we can see, the database is clearly inconsistent as its records show that the employee Gregory was in Zone 1 and in Zone 2 at the same time. If we decide to use facts to represent this information, we obtain the following:

has been("Gregory", "Zone1", "11 : 00", "02/14/2012"),
and has been("Gregory", "Zone2", "11 : 00", "02/14/2012").

This scenario is incorrect because contradictory information is obtained when we represent in a natural way that it is not possible to be in two places at the same time using the following rule:

$\sim$ has_been(Employee, Zone, Time)
$\leftarrow$ has_been(Employee, Different_Zone, Time),
Different_Zone $\neq$ Zone

Then, the set

Π = {$\sim$ has_been(Employee, Zone, Time)
$\leftarrow$ has_been(Employee, Different_Zone, Time),
Different_Zone $\neq$ Zone.,
has_been("Gregory", "Zone1", "11 : 00", "02/14/2012"),
has_been("Gregory", "Zone2", "11 : 00", "02/14/2012")}

is inconsistent.

Security Database

Employee

id	name
1	Gregory
2	Chase
3	Cameron
4	Eric

Security Cameras

employee	zone	timestamp
1	1	11:00 02/14/2012
4	1	09:00 02/14/2012
4	3	11:00 02/15/2012
1	2	11:00 02/14/2012
2	1	09:00 02/14/2012

Fig. 2. A database with logs from security devices.

We have introduced presumptions in the context of DeLP as a device to represent information that is tentative and can be assumed as valid whenever no contradiction arises; that is how we will use the information coming from external sources. Given a database D, we call *operative presumptions* those tentative facts associated with the information stored in D. Formally:

Definition 5 (*Operative Presumption*). Let X be a set of predicates, *pred* the predicate name for some $x \in X$, L an atom of the form $pred(t_1, \ldots, t_m)$, and $\mathbf{D} = \{D_1, \ldots, D_n\}$ a set of databases. An *operative presumption* (OP) for a database D_k and the predicate x is a presumption "$L \prec$" such that there exists a tuple $tup = (q_1^k, \ldots, q_m^k) \in D_k$ with $1 \leqslant k \leqslant n$, where $q_i^k = t_i$ for all i. OPs are also denoted as "$L \prec true$".

The set of all OPs for given sets of predicates X and set of databases $\mathbf{D} = \{D_1, \ldots, D_n\}$ is defined as:

$$OPset_{\mathbf{X},\mathbf{D}} = \bigcup_{i=1}^{n} OPset_{X,D_i}$$

where $OPset_{X,D_i}$ is the set of all OPs for database D_i and every predicate $x' \in X$.

Example 3. Assume we have the database in Fig. 3 with information about certain drugs that were administered to patients to treat an illness affecting them. The database contains the tuples (xavier, pneumonia, penicillin, great) and (jean, pneumonia, penicillin, bad), with the outcomes of the treatment for Pneumonia with Penicillin in some patients. From that information we can obtain the following OPs:

treated(xavier, pneumonia, penicillin, great) $\prec$ true.
treated(jean, pneumonia, penicillin, bad) $\prec$ true.

Unlike the situation of Example 2, the information shown in this example is not contradictory as different patients may have different reactions to the same medicine. But contradiction still could appear in other ways; suppose that we have in our program two conflicting defeasible rules: d-rule (a) states that if a certain drug has been successfully administered to some patient presenting symptoms of a particular disease, then the drug should be recommended for new cases of the same disease; on the other hand, d-rule (b) concludes the contrary, motivated by unsuccessful instances of the drug application. Formally

(a) treat_with(Disease, Drug) $\prec$ treated(_, Disease, Drug, great).
(b) $\sim$ treat_with(Disease, Drug) $\prec$ treated(_, Disease, Drug, bad).

Based on these rules and the retrieved OPs we have reasons both in favor of starting the treatment of pneumonia with penicillin and also reasons against doing that. So, even if we use consistent information from relational databases, contradictory conclusions can be obtained. In these scenarios, defeasible argumentation introduces methods allowing us to obtain coherent answers

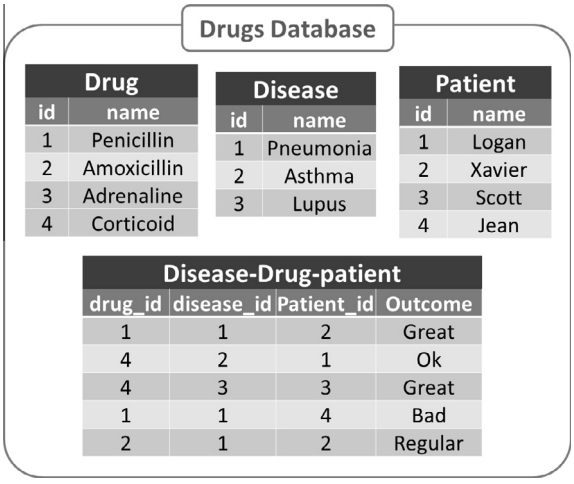

Fig. 3. Database containing information of disease treatments.

enriching our reasoning capabilities, e.g. by giving prevalence to rule (b) over rule (a), from the information available we decide we should not treat pneumonia with penicillin when some reaction is bad.

We extend DeLP programs to include information as OPs obtained from databases; thus, a DBI-DeLP program, or *dbi-delp*, integrates a *delp* as defined in Section 2 and a set Σ of OPs. As we will show in Section 3.3, such OPs are retrieved on demand when required by the DBI-DeLP server for solving a particular query and discarded after the query is solved. Formally:

Definition 6 (*DBI-DeLP Program*). Let $\mathbf{D} = \{D_1, \ldots, D_n\}$ be a set of databases, $\mathcal{P} = (\Pi, \Delta)$ a *delp*, X the set of every predicate in the rules of $\mathcal{P}$. A *dbi-delp* $\mathcal{P}'$ is a triplet (Π, Δ, Σ) where $\Sigma = OPset_{\mathbf{X},D}$ is the set of OPs for $(X,\mathbf{D})$.

Now we describe the process used to answer queries from a *dbi-delp*. In Definition 3 of Section 2 we have outlined how DeLP constructs arguments to solve queries by a backward chaining process. That is, when 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*). Given a strict rule $L \leftarrow L_1, \ldots, L_n$, or a defeasible rule $L \prec L_1, \ldots, L_n$; every literal L_i $(1 \leqslant i \leqslant n)$ in the body of the rule is called Target Goal (TG). The set of all TG is called *TGset*.

The TGs are a key element in DBI-DeLP, as they are the connection between the rules in the program and the records in databases. In the rest of this section we will show how these elements relate to each other.

Next, we introduce an example to clarify how TGs arise by means of the backtracking performed by the SLD resolution. The example also shows the different components of a TG.

Example 4. Consider the *dbi-delp* with the set Δ shown in Fig. 4. Consider the query $\sim$ buy_stock (acme)?; to answer it, the dialectical process can use the rule:

$$\sim \text{buy\_stock}(C) \prec \text{good\_price}(C), \text{risky\_company}(C)$$

There, good_price(acme) and risky_company(acme) become TGs. Also, when the server tries to obtain risky_company(acme), new TGs appear: in_fusion(acme, AC), strong(AC) and closing(acme). As we can see, all TGs have the form *pred* $(t_1,\ldots,t_m)$, where *pred* is a predicate name and $t_1, \ldots, t_n$ is a list of its parameters. For instance, for the TG in_fusion(acme, AC), the predicate name is in_fusion, while acme (a constant) and AC (a variable) are the parameters.

The SLD procedure then tries to prove every TG by using all available knowledge, i.e., all the rules, facts and presumptions in the *dbi-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 presumptions (using condition (a) of Definition 3) 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][1] in the data source is called a Parameter Source (PS). Formally,

Definition 8 (*Parameter Source*). Given a set **D** of available databases $\{D_1, \ldots, D_n\}$, 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 every PS is called *PSS*.

Each potential data source of useful information for a given TG is linked to the corresponding TG through a Pertinence Relation.

Definition 9 (*Pertinence Relation*). Given a set **D** of available databases $\{D_1, \ldots, D_n\}$, let *PSS* be the set of all PS for **D** and *TGset* the set of all TG. The Pertinence Relation PR $\subseteq TGset \times PSS$ is such that if (TG, PS)$\in$ PR then PS is a pertinent source for TG.

We assume that the Pertinence Relation is given as an input to the system; in Section 3.2.3 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 OPs from this source. But before we can actually obtain the presumptions, we need to find out where to look for them. To do so, we use the Data Sources Retrieval function (DSR) which returns, according to the Pertinence Relation, *all* the pertinent data sources for a given TG, i.e., the fields in tables we need to look when searching for support to the TG. The DSR retrieves one set of [database, table, field] triplets, which we refer to as a *data source*, for any source prone to contain information for a TG, or an empty set if the TG cannot be proven using the available databases. Intuitively, every [database, table, field] triplet in a data source for a TG indicates the field, table and database from which we can obtain data for a parameter in the literal TG.

Definition 10 (*Data Source Retrieval*). Given a set **D** of available databases $\{D_1, \ldots, D_n\}$, let *PSS* be the set of every PS for **D**, *TGset* be the set of every TG, and TG $\in$ *TGset*. Let $DS \in PSS$ be a set of PS. Let PR be the Pertinence Relation for *TGset* over *PSS*. The Data Source Retrieval DSR: $TGset \mapsto 2^{PSS}$ is such that DSR(TG)=$\{TG \in TGset : (TG, DS) \in PR\}$.

Example 5. Suppose that we have available the database shown in Fig. 5 and the TG is performs_in(Movie, Actor). Then

$$\text{DSR}(\text{performs\_in}(\text{Movie}, \text{Actor})) = \{\{[\text{movies}, \text{title}], [\text{actors}, \text{name}]\}\}$$

as {[movies, title], [actors, name]} is pertinent to the TG performs_in(movie, actor). In fact, by looking in the title field of table Movie and the name field of the Actors table we can find out if a certain actor took part in the cast of a given film. [2]

[1] To ease reading, we will omit the database for those cases in which the pair [table, field] can unequivocally identify the data source.

[2] Actually, we have to look in the corresponding fields of the SQL JOIN made over the three presented tables.

$$\Delta = \left\{ \begin{array}{rcl} \text{buy\_stock(C)} & \prec & \text{good\_price(C).} \\ \sim\text{buy\_stock(C)} & \prec & \text{good\_price(C),} \\ & & \text{risky\_company(C).} \\ \text{risky\_company(C)} & \prec & \text{in\_fusion(C, AC).} \\ \text{risky\_company(C)} & \prec & \text{closing(C, AC)} \\ \sim\text{risky\_company(C)} & \prec & \text{in\_fusion(C, AC),} \\ & & \text{strong(AC).} \end{array} \right\}$$

Fig. 4. A *dbi-delp* for the stock market.

For example, the queries

$$\text{performs\_in(commando, stallone)?}$$
$$\text{performs\_in(demolitionman, bullock)?}$$

are both answered positively. Notice that the pertinency relation does not guarantee that a given database will in fact support the literal; e.g. we cannot use the database in Fig. 5 to support performs_in(the one, jet li), as there is no tuple in the database indicating that Jet Li performs in the movie The One. Instead, the pertinency relation indicates that the data source is a suitable candidate as it may have the required data (in this case, adding the value the one to the Movies table and jet li to the Actors table and linking them in the Movie-Actor table).

As it can be seen in the previous example, we handle the DSR as an abstract function. In Sections 3.2.3 and 3.3 we show how to instantiate this function using specific information from the databases. 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 *dbi-delp*. This retrieval is made by the application of the Presumption Retrieval Function (PRF). Intuitively, the goal of the PRF is to feed the argumentation process with relevant data obtained from the pertinent data sources.

Definition 11 (*Presumption Retrieval Function*). Let $\mathbf{D} = \{D_1, \ldots, D_n\}$ be a set of available databases, *TGset* be the set of all target goals and *OPset* be the set of all operative presumptions. Let $TG = pred(t_1, \ldots, t_m) \in TGset$, and let *PSS* be the set of all parameter sources. The PRF:$TGset \mapsto OPset$, is such that PRF(TG) = S where

1. $pred(q_1, \ldots, q_m) \prec true$ in S iff there exists a tuple $tup = (q_1, \ldots, q_m)$ in the database $D \in \mathbf{D}$ such that if $ground(t_i)$, then $q_i = t_i$, for all $1 \leqslant i \leqslant n$
2. there exists a $P \subseteq PSS$ such that
   - DSR(TG) = P, and
   - for every $q_i \in tup$ it holds q_i belongs to field F in table T of database D and [D,T,F] $\in$ P.
3. S is $\subseteq$-maximal: there is no set S' of *OPset* such that $S \subsetneq S'$ satisfying (1) and (2) above.

Therefore, the PRF function retrieve database tuples from pertinent data sources with values equal to the corresponding grounded values. For example, consider a database with the tuples (demolition man, stallone), (demolition man, snipes) and (rambo, stallone), where each tuple states that the actor in the first component has appeared in the film associated with the second component. Given the TG performs_in(demolition man, Actor), for the PRF we have

$$\text{PRF}(\text{performs\_in}(\text{demolition man}, \text{Actor}))$$
$$= \{\text{performs\_in}(\text{demolition man}, \text{stallone})$$
$$\prec \text{true}, \quad \text{performs\_in}(\text{demolition man}, \text{snipes}) \prec \text{true}\}$$

unifying the non grounded parameter Actor with both stallone and snipes. As we can see, performs_in (rambo, stallone) $\prec$ true is not in S as the tuple (rambo, stallone) does not match the value required for the grounded parameter.

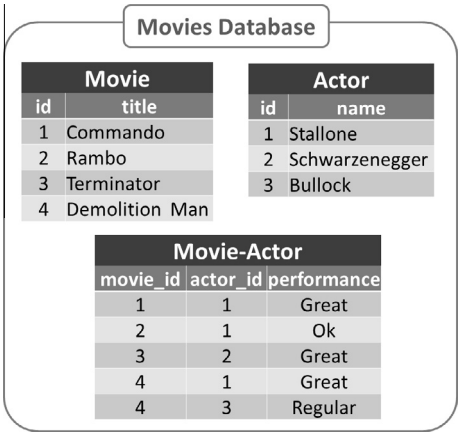

Fig. 5. Database with data about films' casts.

3.2. Implementing the components of the framework

Having introduced the fundamental theory for DBI-DeLP, we move onto its implementation. We use a three-component architecture that captures the behaviors of the retrieval functions used to obtain the pertinent datasources and the relevant data from them.

To integrate DeLP with a database system we need to identify the databases that can be used during the argumentation process. We assume that our database system may involve several databases, which are accessed asynchronously. In running time, the set of databases could change, adding new databases and/or removing existing databases. 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 schemes remain unchanged. To make this possible we introduce a translation layer between DeLP and such schemes. The three component architecture of our framework will facilitate achieving the goals just outlined: the DBI-DeLP Server is in charge of performing the argumentation process, the Domain Data Holder (DDH) is used to store domain knowledge, and the Domain Data Integrator (DDI) recovers data from the domain knowledge to support arguments and feeds it to the argumentation process. Next, we present these components, describing their purpose and how they relate to each other.

3.2.1. DBI-DeLP server

The DBI-DeLP Server component receives DeLP ground queries, then it builds arguments and counterarguments based on its knowledge base, and gives answers and explanations of how they were obtained. We can say that all the system's knowledge is stored in this component as it maintains all the rules and facts of the domain. Also, this component considers OPs that are used to construct arguments (unlike the DDH, which only stores raw, unprocessed data). Two modules are included in the DBI-DeLP server to separate knowledge storage issues from the actual usage of such knowledge: Domain Logic and DeLP Core.

Domain Logic. Through the Domain Logic we capture the domain knowledge available to the system, expressed as a *dbi-delp*. For example, an Argument-based Movie Recommender System [29] using DBI-DeLP could have strict rules such as:

$$\text{child\_restricted}(\text{Movie}) \leftarrow \text{has\_violence}(\text{Movie})$$

and defeasible rules such as:

$$\text{has\_violence}(\text{Movie}) \prec \text{director}(\text{Movie}, \text{tarantino})$$

Dynamically, OPs such as:

$$\text{film\_genre}(\text{pirates\_of\_the\_caribbean}, \text{comedy}) \prec \text{true}$$
$$\text{film\_genre}(\text{pirates\_of\_the\_caribbean}, \text{action}) \prec \text{true}$$

could be added if different categorizations for the film Pirates of the Caribbean are found in the domain data when required by a query execution.

DeLP Core. This module is in charge of computing the final answer by constructing dialectical trees and analyzing them. It receives a query like good_movie(commando)? from a client and tries to build arguments for and against it using the Domain Logic. Finally, it gives the resulting answer, which can be YES if an argument in favor is warranted, NO if an argument against it is warranted, UNDECIDED if neither arguments for nor against it can be warranted, and UNKNOWN if the query includes literals that are not in the program's language [11].

3.2.2. Domain data holder

This component is a massive and potentially contradictory set of domain-related data that provides ground information during the argument building process. In the current version of the framework, data is stored in and accessed from independent relational databases via an Open DataBase Connectivity (ODBC) driver. There is no upper bound to the number of databases included in the DDH, and the addition or removal of a database has no effect on the others; nevertheless, the total knowledge is altered, so if a previous query is issued again after this change the resulting answer may be different. There are no restrictions regarding how tables and fields should be named, or how the database schema should be specified; however, since the server needs to identify which tables and fields are necessary to include in the SQL query to solve a given TG, the configuration of each database regarding the TG should be provided, i.e., the PR for it should be given.

3.2.3. Domain data integrator

As mentioned above, we leave unchanged the representational structures of DeLP and only consult the databases for information. We introduce an intermediate layer such that, given a particular query, it takes the associated information from the database and adapts it in a way that can be used by DeLP's reasoning engine. This layer allows the interaction between the DeLP Server and the DDH by creating SQL queries to retrieve info datasets from the databases during the argument building process.

The DDI transforms datasets into OPs used to build arguments. Before the DDI can retrieve relevant datasets, it needs information about the data sources that are pertinent to the target goals it is trying to validate; i.e., this component is in charge of the execution of the DSR function. As stated before, the PR used by the DSR is an input to the system, i.e., the PR is given by the user. In the implementation of the DSR, we use an auxiliary database, the Predicate Translation Database or PTD, to maintain the PR; this database maintains information about the relation between TGs and pertinent data sources. Note that, from a PS [D,T,F], the component D has a correspondence with some DSN in the Predicates table of the PTD that indicates the database, and the components T and F will correspond with the table and field indicated in an associated record in the Parameters table. Additionally, the PTD has other information needed to perform the execution of the SQL Queries, e.g. the user and password used to login to the DBMS.

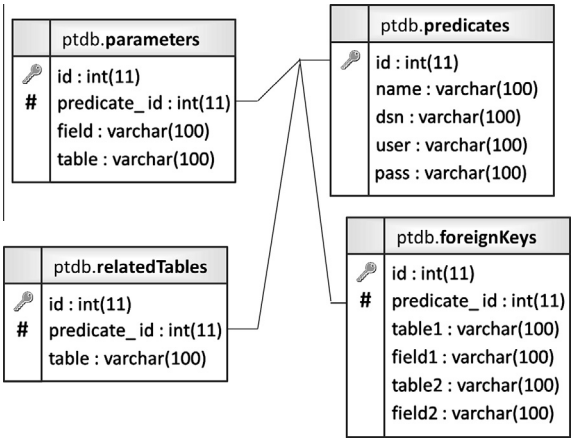

Fig. 6. The predicate translation database schema.

Thus, given a particular TG, the DDI uses the DSR function to retrieve the schema information from the PTD that the SQL query has to follow for every pertinent data source to find relevant data in them. The PTD keeps the four tables shown in Fig. 6:

- Predicates table: maintains the correspondence between predicate names and data sources.
- Parameters table: maintains the equivalence between a predicate's parameter and a pair (table, field) in some database.
- RelatedTables table: keeps information about the tables that take part in the SQL JOINS needed to obtain information about a particular predicate.
- ForeignKeys table: maintains a list of the pairs (primaryKey, foreignKey) for the SQL JOINS.

3.3. Processing of queries in DBI-DeLP

After describing the components of the framework, we show how they relate to each other so data from relational databases can be exploited to provide support for arguments built in the dialectical process behind solving queries posed to the system. We introduce the theoretical foundations used by the process, and present the algorithms for their implementation. Finally, we show an example of how the integration of the components allows the use of data from relational databases to answer queries.

Fig. 7 shows a general schema of the process of obtaining the supporting data each time the DBI-DeLP Server receives a query. It is important to remark that the process of answering a query is considered as part of a closed transaction in the sense of database theory [30,31]. Essentially, it involves building arguments for and against the literal queried using the *dbi-delp* of the Domain Logic, and when it is necessary to request data stored in a database the DDI searches the DDH for useful tuples that will be returned as OPs to be used by the DeLP Core.

We developed algorithms implementing the DSR and the PRF functions that have been defined. For the DSR function, the PTD contains all pertinent data sources for each predicate we need to retrieve from the databases; thus, the function obtains from the PTD all the parameters necessary to connect to the databases in the DDH, plus the names of the fields and tables that are necessary, and the information for executing SQL joins to solve the query (see Algorithm 1).

Algorithm 1 (*Data Source Retrieval Function*).

1: **function** DSR (Predicate Name *predName*):*DatasourcesList*
2: Execute a SQL Query to the PTD of the form
3: "SELECT Id, DSN, User, Pass FROM predicates WHERE name = *predName*"
4: **for each** DSN obtained **do**
5: dsn_id ← present DSN Id
6: ***fieldsToRetrieve*** ← Execute a SQL Query in the form
7: "SELECT Table, Field FROM parameters WHERE predicate_id = dsn_id"
8: ***joinTables*** ← Execute a SQL Query in the form
9: "SELECT Table FROM relatedTables WHERE predicate_id = dsn_id"
10: ***joiningFields*** ← Execute a SQL Query in the form
11: "SELECT Tables, Fields FROM foreignKeys WHERE predicate_id = dsn_id"
12: *DatasourcesList*[*i*] ← [Id, DSN, User, Pass, fieldsToRetrieve, joinTables, joiningFields]
13: **end for**
14: **return** *DatasourcesList*
15: **end function**

Having defined how we obtain the pertinent datasources for the TG we are trying to prove, we introduce the function that retrieve relevant data from the DDH. Operationally, the PRF needs to execute SQL queries over the databases in the DDH and format the resulting datasets to make it processable by the DeLP Core; for that they use three other functions:

- *obtainInstantiatedParameters*: this function receives a list of parameters in a TG and returns the ground ones; e.g. from director(Movie, tarantino) it will return the list [tarantino], and from film_genre(pirates of the caribbean, comedy)it will return [pirates of the caribbean, comedy].
- *obtainInstantiatedFields*: this function takes a list of fields and a list of parameters and returns those fields corresponding to instantiated parameters; e.g. receiving [[table.field1, table.field2], [Movie, tarantino]] it will return [table.field2].
- *generateOperativePresumption*: this function receives a predicate name and a list of values and returns an atom with the predicate name and the values as parameters; i.e., receiving [film_genre, ['Game of Thrones', 'Drama']] returns the OP film_genre('Game of Thrones', 'Drama')$\prec$true.

Having outlined the auxiliary functions used in the algorithm, we turn to the implementation of the PRF used by DBI-DeLP to obtain OPs for a TG (see Algorithm 2). Note how the search for a TG is done on each DSN listed as a possible support data holder. Also observe that a part of the query could be statically built given the structural information of the data sources. Nevertheless, the *WHERE* part of the query depends on which parameters are instantiated in the TG, leading to dynamically build the query considering these variable conditions. Hence, given this dynamic nature of TGs it is not possible to directly maintain in the PTD a mapping between a predicate and a static SQL query, even though this would have been simpler.

Algorithm 2 (*Presumption Retrieval Function*).

1:**function** PRF (Target Goal L_i):*operativePresumptionsList*
2: Decompose L_i into its predicate name *predName* and a list of parameters $t_1, \ldots, t_n$
3: ***instantiatedParameters*** ← obtainInstantiatedParameters($t_1, \ldots, t_n$)
4: ***DataSourcesList*** ← DSR(*predName*)
5: **for** *i* **from** 1 **to** *length*(*DataSourcesList*) **do**
6: ***whereFields*** ← obtainInstantiatedFields(*fieldsToRetrieve*$_i$, $t_1, \ldots, t_n$)
7: Connect to database indicated by DSN$_i$ using user User$_i$ and password Pass$_i$

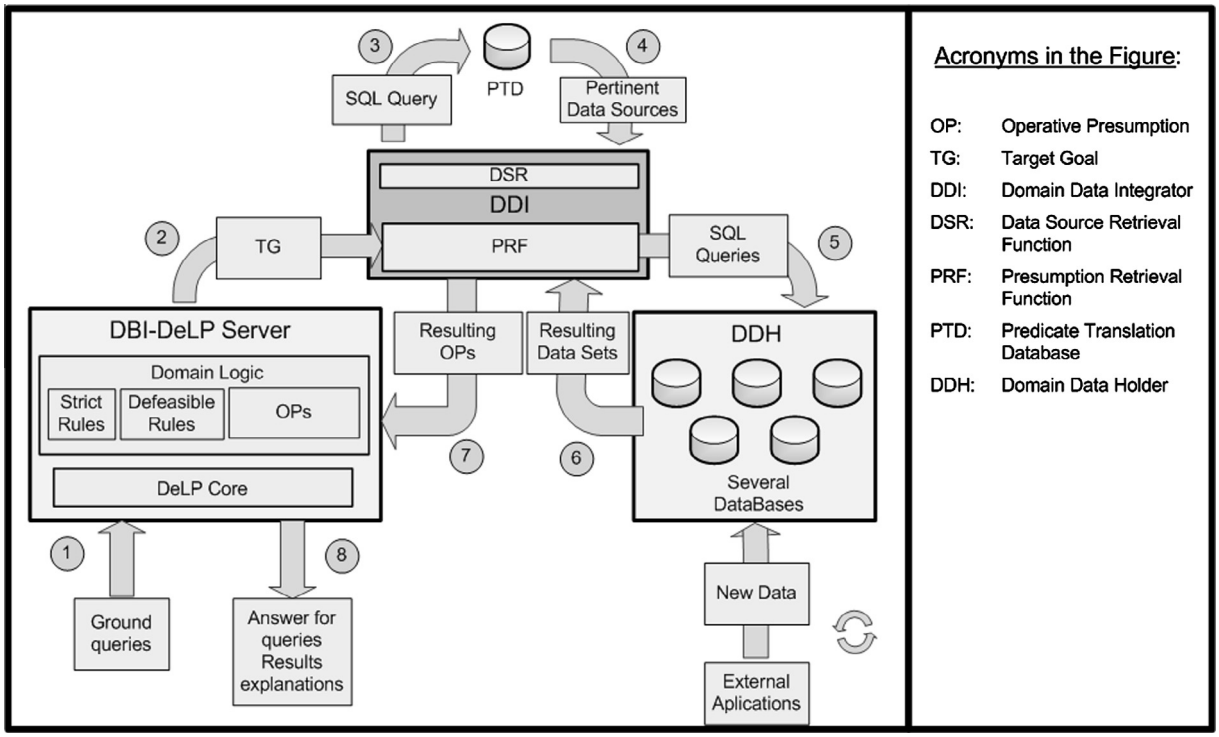

Fig. 7. The DBI-DeLP argumentation process.

8: results ← Execute a SQL Query in the form
9: "SELECT ***fieldsToRetrieve***$_i$ FROM **joinTables**$_i$ ON ***joiningFields***$_i$
10: WHERE ***whereFields=instantiatedParameters***"
11: **for each** result res_i obtained **do**
12: operativePresumptionList[i] ← generateOperativePresumption(*predName*, res_i)
13: **end for**
14: **end for**
15: **return** *operativePresumptionsList*
16: **end function**

A special case of TG search can be performed when all the parameters in the TG are grounded. In this case, we do not need to retrieve every tuple supporting the TG: as it is fully grounded, there is just one possible OP alternative. Thus, we just need to find a tuple fulfilling the search conditions. To capture this particular behavior, we could introduce two modifications in the algorithm. First, we can break the loop in which every DSN is searched as soon as the first result appears. Also, the SQL query on lines 9–10 of Algorithm 2 may be changed to

SELECT ***fieldsToRetrieve*** FROM ***joinTables***
ON ***joiningFields***
WHERE ***whereFields*** = ***instantiatedParameters***
LIMIT1

so that the DBMS stops the search after the first result.

3.4. A complete query processing example

With the framework introduced, we will demonstrate the complete query solving process. For the sake of example, we will show next how two *dbi-delp* that solve queries w.r.t. the same information in the proposed framework, may end up obtaining different conclusions.

Let us consider an example where we have a moderator and several agents related to the movies domain that will determine if a certain movie is recommendable for a particular user. We capture the recommending behavior of the agents using two *dbi-delp* following different approaches; for instance, we will have a program A representing a movie critic, and a program B representing a family and childhood preservation service. These two programs clearly have different approaches to recommend a movie. In DBI-DeLP, these differences can be expressed in terms of strict and defeasible rules. Nevertheless, both programs will access the same database, i.e., they have access to the same information about the movies, but they will use different criteria considering different aspects. The programs are shown in Fig. 8 and they represent the rules for the movie critic (A) and the organization (B). While we present a reduced example with a few rules it is clear that in real-world applications the program will be larger.

As for the Σ set, an extract of the data is depicted in Fig. 9. We can see that the DDH has two databases: the Movies and Series database stores information about films, while the System database stores information about registered users of the recommender.

Here we can see one advantage of using database stored information to develop these programs; otherwise, if we want them to have updated information about several movies, we will need to include manually in every agent the information for every film (consider that data showed in Fig. 9 is only an extract, but obviously the DDH can store information about millions of films).

Finally, the PTD needs to be set up as shown in Fig. 10 to properly translate the queries sent by the DeLP core into SQL queries performed on the databases associated with the DDH.

So, given the presented scenario, the agents can build arguments and give the moderator their vote on whether or not to recommend the film to the user. Assume that the moderator receives the query recommended(''Game of Thrones'', ''Robb'')?. Then, as said before, it will send the query to every agent in the system. For the purpose of this paper we will focus on the process carried out by the DeLP Server that has the program A in its Domain Logic to determine if it recommends the film or not. Next, we give a description of the steps that take place in the DBI-DeLP server when solving a query. The steps in the example are presented in correspondence with the number of the steps in the DBI-DeLP process shown in Fig. 7.

1. Query recommended (''Game of Thrones'', ''Robb'')? is received by the DBI-DeLP server.
2. DeLP core searches for recommended(Film, User) as head of a rule, finding

$$\text{recommended(Film, User)} \prec \textit{genre}(\textit{Film}, \textit{Genre}), \textit{likes}(\textit{User}, \textit{Genre}), \textit{high\_rating}(\textit{Film}).$$

next the server tries to satisfy the first literal in the body, i.e., genre(''Game of Thrones'', Genre). There is no rule available to determine

$$\Pi_A = \left\{ \begin{array}{rcl} \text{high\_rating(Film)} & \leftarrow & \text{rating(Film, Excellent).} \\ \text{high\_rating(Film)} & \leftarrow & \text{rating(Film, Very good).} \end{array} \right\}$$

$$\Delta_A = \left\{ \begin{array}{rcl} \text{recommended(Film, User)} & \prec & \text{genre(Film, Genre), likes(User, Genre), high\_rating(Film).} \\ \sim \text{recommended(Film, User)} & \prec & \text{genre(Film, Genre), likes(User, Genre), bad\_cast(Film).} \\ \text{bad\_cast(Film)} & \prec & \text{performance(Film,Actor,"bad").} \\ & & \sim \text{performance(Film,Actor2,"great").} \end{array} \right\}$$

$$\Pi_B = \left\{ \begin{array}{rcl} \text{child\_restricted(Film)} & \leftarrow & \text{observation(Film, "Nudity").} \\ \text{child\_restricted(Film)} & \leftarrow & \text{observation(Film, "Violence").} \\ \text{child(User)} & \leftarrow & \text{age(User,Age), Age} < 13. \\ \text{high\_rating(Film)} & \leftarrow & \text{rating(Film, Excellent).} \end{array} \right\}$$

$$\Delta_B = \left\{ \begin{array}{rcl} \text{recommended(Film, User)} & \prec & \text{high\_rating(Film), gender\_recommended(Film, User).} \\ \sim \text{recommended(Film, User)} & \prec & \text{high\_ rating(Film), gender\_recommended(Film, User),} \\ & & \text{child(User), child\_restricted(Film).} \\ \text{gender\_recommended(Film, User)} & \prec & \text{gender(User, "Male"), observation(Film, "Violence").} \\ \text{gender\_recommended(Film, User)} & \prec & \text{gender(User, "Male"), observation(Film, "Epic").} \\ \text{gender\_recommended(Film, User)} & \prec & \text{gender(User, "Female"), observation(Film, "Romance").} \end{array} \right\}$$

Fig. 8. Two *dbi-delp* (Π_A, Δ_A) and (Π_B, Δ_B) for movie recommendation.

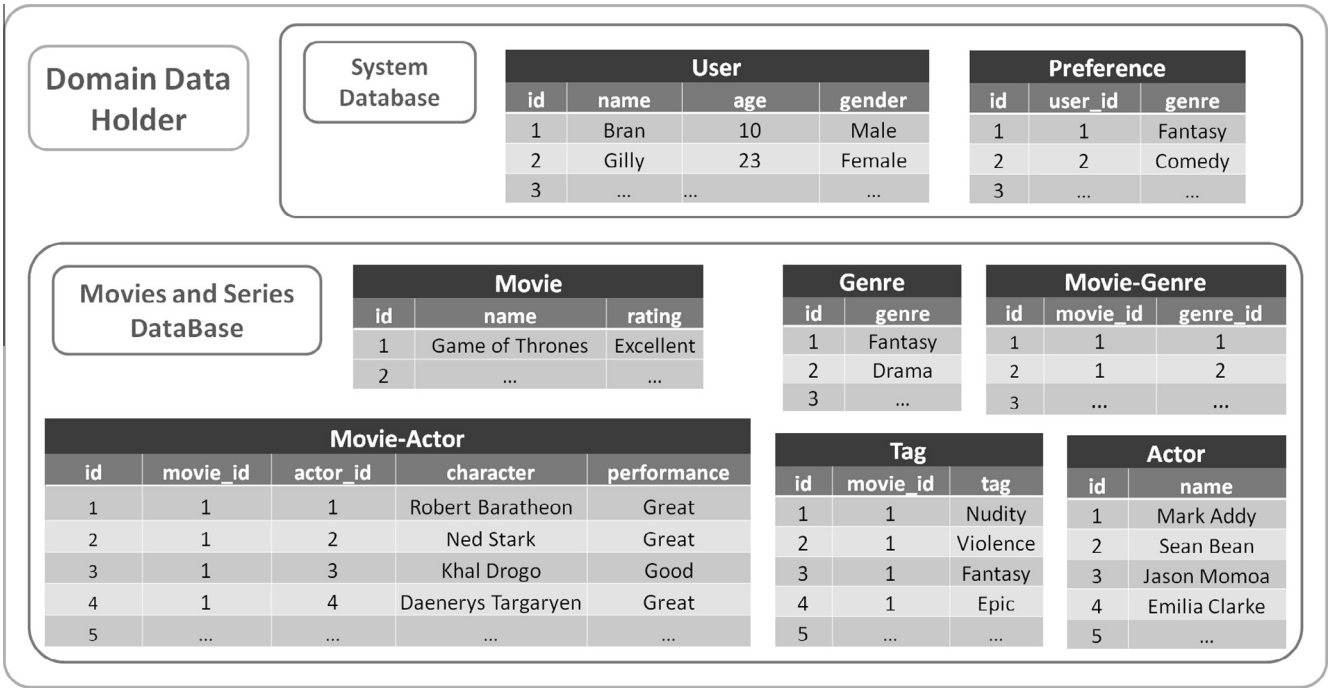

Fig. 9. The data maintained in the DDH databases.

the genre of the film Game of Thrones, so the DDH is searched by executing PRF(genre(''Game of Thrones'', Genre)).

3. DDI takes genre(''Game of Thrones'', Genre) and decomposes it in its predicate name genre and the list of its parameters [''Game of Thrones'', Genre]. Then the DDI looks in the PTD for information about pertinent data sources to the TG using DSR(genre).
4. The DBMS that manages the PTD answers those queries sent by the DDI. That way the DDI knows which tables to look for information, and how the fields are named. For this example the result of DSR(genre) is that the Movies and Series database may contain information about the genre predicate (and which tables and fields we need to search).
5. Now that it knows where to look for information, the PRF executes DSN-specific SQL queries to each DSN retrieved. In this example, it will execute

```
SELECT movie.name, genre.genre FROM movie
  JOIN movie − genre ON
    movie.id = movie − genre.movie_id
  JOIN genre ON movie − genre.movie_id = genre.id
  WHERE movie.name = Game of Thrones
```

to the Movies and Series database. Notice that only grounded arguments are used for WHERE conditions by the DDI.

6. The DBMS answers the query with the genres listed in the database for Game of Thrones. For the given scenario, the list

```
[row("GameofThrones", "Fantasy"),
row("GameofThrones", "Drama")]
```

is the answer. DDI takes that result and formats them as the OPs

```
genre("GameofThrones", "Fantasy") ≺ true,
genre("GameofThrones", "Drama") ≺ true
```

7. Once the PRF finishes its execution, all the results obtained are sent to the DeLP core as a list of OPs. The DeLP core unifies the second argument on each result with the second argument in likes(User, Genre) and continues the procedure.
8. We get likes(''Bran'', ''Fantasy''), because is in the System database, high_rating(''Game of Thrones'') is obtained from the related rule in Π_A high_rating(Film) ← rating(Film, Excellent), and the Movies and Series database states that the rating of Game of Thrones is Excellent. The DeLP core attempts to build a counterargument trying to find support for the head of the defeasible rule below with the instantiation Film = ''Game of Thrones'' and User = ''Bran'',

$$\sim\text{recommended}(\text{Film}, \text{User}) \prec \text{genre}(\text{Film}, \text{Genre}),\ \text{likes}(\text{User}, \text{Genre}),\quad \text{bad\_cast}(\text{Film}).$$

The TGs genre(Film, genre) and likes(User, Genre) are satisfied as before; but the server cannot obtain bad_cast(''Game of Thrones'') as there is no actor in the film with a bad performance. Since a counterargument against a sub-argument cannot be built, the DeLP Server answers YES to the query, stating the film Game Of Thrones is recommended for user Bran.

A similar process can be followed by the DeLP Server with the program B in its Domain Logic, but with a different result. In this case, based on the sets Π_B and Δ_B, the program gives a negative recommendation to the user Bran for the film because he is not mature enough to watch it, as the system database states that he is ten years old and the Movies and Series database states that there are nude scenes and violence in Game of Thrones.

4. Time efficiency and complexity results

We have seen how defeasible argumentation frameworks like DeLP can be combined with relational databases to perform argumentation over massive amounts of data. Moreover, in Section 3.4 we have outlined how a DBI-DeLP based movie recommender system can solve a query based on ground information available from relational databases. A major issue in such a system is to satisfy reasonable response-time requirements, specially in interactive or multi-agent environments, where such time constraints play an important role. Consequently, to determine whether DBI-DeLP

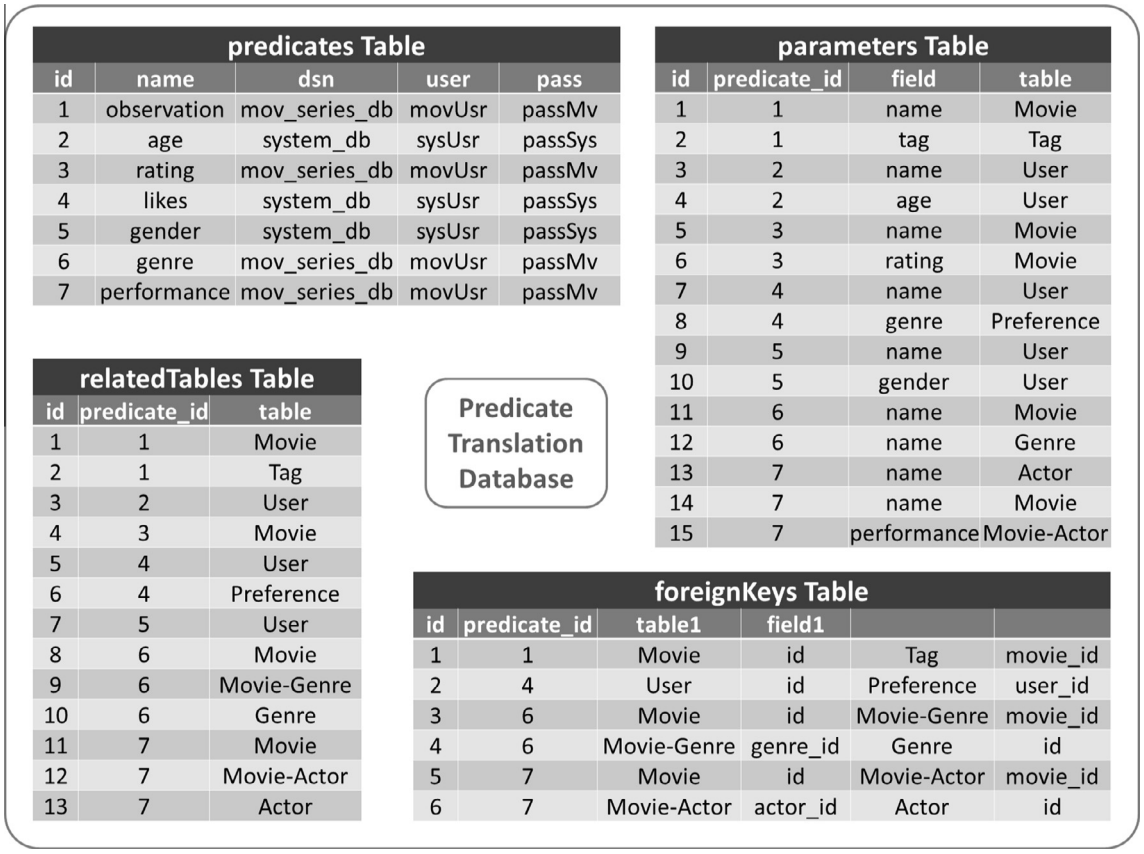

Fig. 10. The setup of tables in the predicate translation database.

is suitable as a tool for supporting the development of such applications, an empirical analysis of complexity and efficiency is required. Generally speaking, the process behind the use of relational databases to support arguments involve two big aspects. First, relevant data needs to be obtained from the available databases. Second, that data must be used to build arguments for a particular query. In this section we introduce analyses of time efficiency and complexity of these aspects to determine if the proposed approach is suitable to develop real-world applications.

4.1. Empirical analysis

In a DBI-DeLP based application there are two main aspects to consider regarding its time efficiency: how much time is consumed searching the DDH for supporting information, and how much time the proof procedure used by the DeLP Core takes to determine which argument prevails (and thus which is the final answer for a given query). Since these two aspects are influenced by different elements, we have conducted experiments that focus on each aspect, as discussed next. For these experiments, we have created a dataset using randomly generated data over the structure of three related tables. The number of generated records depends on each particular experiment, as it will be shown later. The rules and queries used are such that they relate elements in the data stored in the databases or in the program, according to each particular experiment. Regarding the experimental environment, we used a virtual machine with the following configuration:

- Operative System: Windows 7 32 bits
- Processor: Intel Core i5-2410 M @2.3 Ghz
- Memory: 1 Gb DDR3 1333 Mhz

4.1.1. Presumption search for target goals efficiency

Consider the query genre(''Game of Thrones'', Genre)? in the example of Section 3.4. Queries for a TG like this one will be common in any DBI-DeLP based application. Thus, a great part of the total time the application dedicates to solve a query is spent searching the DDH for relevant data by the PRF. So, this DDH search by the PRF is a crucial aspect in the overall performance of the system.

There are two main aspects that we must consider in the experiment. First, the number of tables involved in the SQL query is significant, as it determines if SQL JOINS among tables are needed to solve the query or not. Second, the other variable of main concern is the number of records in tables in the DDH: a search in a table with 10,000 (10 K) records differs from another in a table with 10,000,000 (10 M) records. We focus on these two aspects in our experiments. Nevertheless, there are other aspects that impact in the time consumed by retrieving relevant data from databases, although to a lesser extent, e.g. how many of the parameters of a TG are grounded, as this limits the number of results obtained allowing for a faster reply from the DBMS.

To measure the execution time of the function used to search the DDH for argument supporting data, the same query was launched while the number of records of tables in the DDH was increased. Tables 1 and 2 show the time in which a single search of a TG is completed by DBI-DeLP as the number of records in the searched database increases. For the experiment in Table 1 we have used a fully-grounded TG. In the table we can see that SQL Queries take comparable time whether they are executed in an external application such as MySQL Query Browser or in DBI-DeLP. Thus, we can see that most of the time needed to obtain OPs is consumed by the DBMS solving the MySQL query, with virtually no

overhead introduced by the processing of resulting datasets by DBI-DeLP.

As for the second experiment, Table 2 shows the time consumed obtaining results for a TG with two parameters where the first one is grounded and the second one is not, like genre(''Game of Thrones'', Genre)?, when three tables are involved in solving it.

Based on the performed experiments addressing the efficiency of the search of relevant data by the PRF, the results obtained seem to suggest that the search for support in the DDH is in line with the time constraints found in most real-world applications.

4.1.2. DeLP core proof procedure efficiency

Once obtained the information needed to support arguments, we need to analyze the time used by the DeLP Core to establish the answer for a received query.

As we have stated before, the proof procedure depends on the construction of dialectical trees. Nevertheless, instead of measuring the time it takes to construct the entire dialectical tree, for this experiment we have chosen to measure the difference between DeLP and DBI-DeLP in the time needed to construct a single argument. There are two reasons for such a choice. On the one hand, the time needed to construct the dialectical tree depends directly on the number of arguments in it, making it a crucial variable. On the other hand, the only important difference between DeLP and DBI-DeLP regarding the efficiency of the proof procedure is how they differ in the task of constructing arguments; the rest of the dialectical process is the same in both approaches, i.e., the cost of building a dialectical tree in both approaches is the same, provided that we discriminate the time needed to construct arguments. Consequently, to establish differences in the time used for the proof procedure in both approaches in the context of massive amounts of information we contrast the time it takes to build an argument in our approach against the time it takes to build the same argument in a standard, non-database DeLP system; since the difference in efficiency for that construction can be extrapolated to the difference in efficiency concerning the dialectical tree construction and analysis.

The main objective of this experiment is to analyze if our combined method using argumentation-based inference with external databases as provider of support is more suitable (in the context of real-world environments, where massive amounts of data are used) than the standard approaches used to develop RBAS where data is directly encoded in the program.

Table 1
Execution time for a PRF over a fully-grounded TG without the need for SQL JOINS in an unindexed table.

NR	TDBI-DeLP	TMySQL
10 K	0.0189 s	0.0116 s
100 K	0.0381 s	0.0138 s
500 K	0.1469 s	0.1174 s
1 M	0.274 s	0.3094 s
5 M	1.2439 s	1.7238 s
10 M	2.465 s	3.0939 s

1 K = 1000, 1 M = 1,000,000, NR = Number of records, TDBI-DeLP = Execution time in DBI-DeLP, TMySQ = Execution time of SQL in MySQL Query Browser.

Table 2
Execution time for a PRF with SQL Joins between 3 indexed tables.

No of records (Tables T1 - T2 - T3)	Exec. time
50–500–50 K	0.081 s
250 K–2.5 M–250 K	0.2129 s
500 K–5 M–500 K	1.0369 s
1–10–1 M	2.3859 s

The experiment is designed as follows: we have two instances of defeasible logic programs that can support the same arguments; but, while one of them has all the information needed encoded as presumptions *in* the program (a *delp*), the other has the same information externally supported in a database (a *dbi-delp*). That is, the entire set Σ of the *dbi-delp* is included in the *delp* in the set Δ. Then, we pose the same query to these programs so the same argument is built, and we measure the time that takes to built this argument in each case.

The obtained results are presented in Table 3. As shown in the table, we made the experiment for different number of records (or encoded presumptions in the standard approach), and we distinguish two cases:

- The best case, where the data needed to support the argument is the first record in the database, and the first presumption in the program.
- The worst case, where that data is found after searching all the available knowledge.

We can see how the time needed to build an argument varies while the number of records (or encoded presumptions) increases. A graphic view of these results is visualized in Fig. 11.

From these empirical results it can be seen that, on the one hand, in the rarely occurring best case the use of facts in the program is faster than the use of external databases. On the other hand, when analyzing the worst case it can be seen that the database approach behaves considerably better than the classical approach. Moreover, in can be clearly seen from Fig. 11 that while our approach has almost no variation when the number of records increases, the other approach grows linearly.

The intuition behind these results is that, while we let the proof procedure to be in charge of the structure of arguments (i.e., the use of rules and derivations to obtain conclusions) as it is its main function; we pass the responsibility of finding support for the use of these rules to the DBMS. From the experiments becomes apparent that for (very) small amounts of data the approach in which the data is stored directly in the program is more efficient; this is due to the overhead in the interaction between DeLP and DBMS. However, as the amount of data increases, this overhead is palliated by the better performance on data search provided by the DBMS procedures w.r.t. the search performed by DeLP. This is because the DBMS are specialized in managing large amounts of data, i.e., data structures and procedures used by DBMSs to store and handle data are more efficient than the ones used to manage data when this data is directly included in the program.

4.2. Running-time analysis

To complete the study of the implementation of the DBI-DeLP framework we want to study the tractability of the process that

Table 3
Time needed to built an argument.

No of records	WC_1 (ms)	BC_1 (ms)	WC_2 (ms)	BC_2 (ms)
100 K	196	130	40	20
300 K	50	130	91	20
500 K	150	110	140	20
750 K	160	176	181	20
1 M	190	90	250	20
1,25 M	113	70	330	30
1,5 M	150	130	411	31
2 M	270	120	511	30
2,5 M	166	150	681	41
3 M	186	186	771	40

WC_1 = Worst Case (using DB), BC_1 = Best Case (using DB), WC_2 = Worst Case (without using DB), BC_2 = Best Case (without using DB).

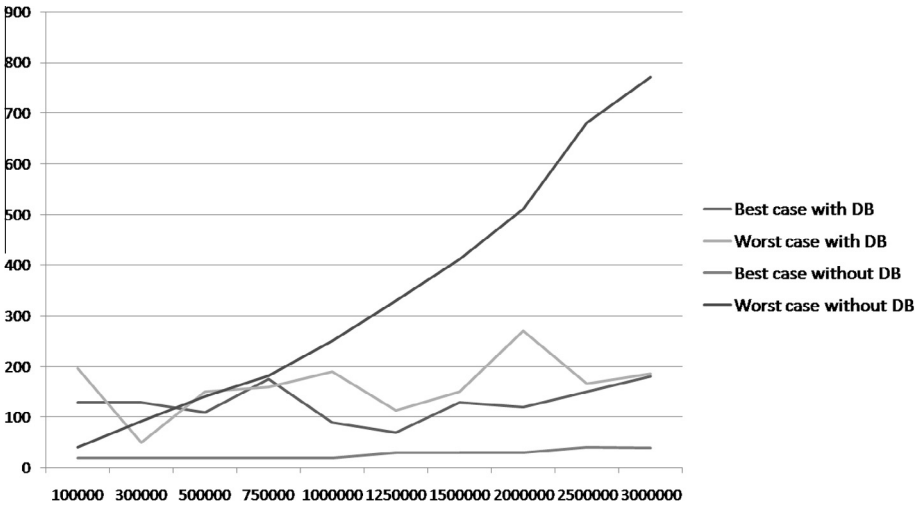

Fig. 11. Efficiency of the proof procedure.

it will use to build arguments and solve queries; again, this can be separated in two parts: the cost of searching for useful data in the DDH and the analysis of how arguments are built based on this data (and the rest of the Domain Logic). In this work we have only conducted the study of the first part, i.e., the function used to find useful data for a TG in the databases that integrate the DDH. A study on the complexity of the process used to built arguments by DeLP (and DBI-DeLP) can be found in [32]. The complexity analysis for the PRF used in DBI-DeLP, using the proposed implementation, follows:

1. Decompose an atom in its predicate name and its parameter list and obtain the instantiated parameters in the predicate; both actions are dependent on the number p of parameters, so both are in $O(p)$.
2. Then, the DSR function is used to find information about all databases having useful data for the predicate being searched. The complexity of the DSR depends on the number m of rows on the predicate's table of the PTD because the operations in the DSR function are executed as many times as the number of relevant databases found (for the TG) in the PTD. In particular, these operations are executed m times since, in the worst case, every entry in the predicate table of the PTD is related to the TG. Next we outline the complexity of the DSR.
   (a) The first query on the loop is for finding the parameter source [database, table, field] corresponding to each parameter. To do this it is necessary to look in each of the k entries in the parameters table. The pairs corresponding to the instantiated parameters are put aside to be used as conditions for the SQL query. This is also in $O(k)$. Some presumptions are formed using information retrieved from just one table in a database, but for other presumptions two or more tables need to be related through a join to retrieve them. Therefore, it is necessary to maintain information about the tables and the key-fields involved in the join.
   (b) The second query is used to obtain all tables that will be involved on the search in the corresponding database; thus, the time for this depends on the number l of entries on the relatedTables table.
   (c) Next, DSR makes a query to retrieve the key-fields pairs on which the join is made. It is in $O(r)$, being r the number of rows on the foreignKeys table.
3. The following query is the one made using the obtained query format information, to retrieve from the database the information that corresponds to the searched TG. The time spent on this is in the order of the product of the entries of each table on the join for the query. Suppose we have a query with a join of s tables, each one with T_i rows; then, the time is in the order of $\prod_{i=1}^{s} T_i$.
4. The retrieved tuples are transformed into OPs and added to the OPs' list, with a cost on the order of the number of retrieved tuples, this is $O(v)$.
   Finally, all OPs retrieved from all databases are sent to the DeLP Core.

Since steps (a)–(c) of the DSR function and the steps (3) and (4) of the PRF are executed for every database that contains information about the predicate, and as each database has a different configuration, the time taken in the loop is different in each iteration. Assuming the worst case, for a predicates table with m entries, there will be m databases containing information for a predicate; so, for steps (a)–(c) there will be in each one m different orders, one for each database. For example, considering the time for all the databases, step (a) running-time will be in the order of $\sum_{j=1}^{m} n_j$.

In the worst case, the total time demanded for the search of presumptions about a predicate is

$$O(2 * p + m + \sum_{j=1}^{m}(k_j + l_j + r_j + \prod_{i=1}^{s} T_{j\ i} + v_j))$$

Now, let n be the number of entries of every table on the DDH. In such case, all parameters m, k_j, l_j, r_j, T_{ji} and v_j will be replaced by n, and the running-time will be summarized into

$$2 * p + n + \sum_{j=1}^{n}(n + n + n + \prod_{i=1}^{s}(n) + n),$$

which is in $O(p + n^{s+1})$, 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.

5. Conclusions and related work

Integration of Knowledge-Based Systems with large repositories in the form of databases is been widely acknowledged as central in the development of such systems in a real-world scale [33,34]. Thus, the problem of making inferences from large repositories of data is one that has been addressed over the years by various approaches, e.g. Data Mining [35,36], Machine Learning [37], or Deductive Databases [38,39]. In particular, the latter stands for systems where knowledge encoding is twofold: on the one hand, the extensional database consists of data (or facts) usually encoded by

a predicate with constant arguments in relational databases; on the other hand, there are rules stating possible relations among facts in the extensional database. This approach is known as "loose coupling" [40]. Through the years, there have been many efforts in this direction in the area of Logic Programming and Databases, e.g. by interconnecting Prolog systems to relational databases [38,41–44], in an approach that has some similarities with the one presented in this paper. In this area, Datalog [45] has become the *de facto* standard for the development of deductive databases. As in the approach proposed in this paper, deductive databases infer new facts based on this encoded knowledge.

Nevertheless, there are several known issues in deductive databases, mainly related to the presence of inconsistency among its inferences. A possible solution to this is to change the inference engine to one that can deal with conflicting conclusions, like Defeasible Reasoning does.

Accordingly with the approach of integrating logic engines with large repositories of data like relational databases to infer new information, in this paper we have proposed an integration of defeasible reasoning with database technologies. We have shown how DeLP can be combined with relational databases to carry out argumentation processes over massive amounts of data. We have extended the notion of a DeLP program to include information coming from databases. Also, we have formalized one function used to find suitable information sources for a particular query from the universe of available databases, and another one that extracts that information and adapts it to be used in the inference procedure. Regarding implementation of the concepts, we have introduced a three component's architecture for the framework and the interaction among them. Additionally, we have presented algorithms that instantiate the functions in the framework suiting the presented architecture.

This approach helps to spread the application of RBAS since other systems may provide input data to our framework without requiring a complex interface. This can lead to definitions of new architectures for Argument-based Recommender Systems (see [29]), as well as Decision Support Systems (e.g. [46]). Moreover, in a Multi-agent System setting, databases may store *community knowledge* that agents with argumentative capabilities [47] can share and use for their reasoning process. In particular, a recent line of research for Argument-Based Recommender Systems has been started where DBI-DeLP serves as the development framework. A first approach in this direction can be seen in [48]. Such recommenders are being developed to consume data simultaneously from several real-world datasets, e.g. the MovieLens [28] and Internet Movie DataBase [27] datasets, stored in relational databases.

We have empirically studied the system and presented a running-time analysis of the processes used by DBI-DeLP to obtain argument supporting data from relational databases, comparing also the time needed to build an argument using a classic internally encoded facts approach and our proposal with external databases. The results obtained seems to show that the use of relational databases to support massive defeasible argumentation processes is more efficient than the approach where data is directly included in the program. The difference in efficiency between both approaches is mainly related to the fact that in our approach the responsibility for finding support to build arguments partially lays in the DBMS search engine managing the source of the information which are usually better suited for these tasks.

The obtained results suggest that it is possible to use this approach to develop intelligent systems which can be used in massive information environments while still meeting reasonable time constraints. Thus, DBI-DeLP seems to be a suitable framework to develop real world knowledge-based systems combining the knowledge representation provided by logic programming with the ability of defeasible argumentation to model argument-based inference procedures.

There have been other approaches to integrate relational databases with defeasible argumentation systems, although in different directions from ours. In [49], the authors present a protocol called PADUA that supports two agents debating a classification by offering arguments based on association rules mined from individual datasets. This research focuses on the use of association rules which are mined from the databases, using argument-based dialogs to classify examples; in contrast, our approach is focused on the conceptualization of relational databases as massive information sources for solving queries in an argumentative setting. The PADUA protocol was later generalized to the PISA approach [50], which involves solving classification problems by pooling information from several agents. The PISA approach proposes an *argumentation from experience* paradigm, whereby individual agents argue for a given example to be classified with a particular label according to their local data and arguments are generated dynamically in the form of classification rules.

In [51], the authors present an argument-based framework called A-MAIL to conceptualize inductive learning in a multi-agent setting. Arguments are given by examples and rules (akin to the presumptions and defeasible rules in DeLP) are automatically generated from repositories of data. Based on this approach, this proposal was later generalized to a defeasible reasoning model of inductive concept learning [52]. In contrast, our approach does not deal with the problem of generating new knowledge from existing databases (nor performing any kind of inductive reasoning using machine learning techniques), being rather focused on obtaining a suitable integration of relational database concepts for performing argumentation efficiently.

In [53], the problem of using defeasible reasoning in a massive data repository is addressed, but instead of using databases storing information for supporting conclusions, their repository is the Web, more specifically the Semantic Web. The paper presents a complete system, also including a study of its performance showing that the time consumed by the system is similar to the time obtained in DBI-DeLP. Recently, several large-scale domain-dependent datasets have been released, providing additional motivation for the development of the framework proposed in this paper. For example, as mentioned before, for a movie recommendation scenario websites like the Internet Movie DataBase [27] and MovieLens [28] provide public datasets (supported by relational databases) with users' information and movies they like. Thus, using databases we probably will have access to the same repositories as those accessed by the system in [53] with no loss in efficiency, giving further reasons to choose a defeasible argumentation/relational databases architecture in the development of new Knowledge-Based Systems.

Another work that focuses on the use of argumentation over the massive repository that is the Web is the one by Janjua et al. [3], where they develop a formal framework for Web-based IDSS (Intelligent DSS) for reasoning over incomplete and conflicting information that is based on DeLP as the argumentation engine. The main focus of that work is in knowledge integration in IDSS scenarios, and provide also mechanisms to make the results obtained by the framework shareable, by means of the Argument Interchange Format (AIF) [54]. Clearly their approach differs to ours, but both systems can complement each other. For instance, they acknowledge that certain kinds of Web-based DSS have their key functionalities supported be legacy systems working with databases, where a framework like the one presented in this paper may help in the integration. Moreover, the framework presented in their paper, Web@-KIDSS, stores certain information for profiling in databases supported in MySQL. Thus, the database retrieval mechanisms provided by DBI-DeLP surely will be proven useful in such environment.

Another area where argumentation frameworks are supported by real-world data stored in databases is Legal Case-Based Reasoning (LCBR) [55–57]. In this area, several argumentation-based

approaches were developed in the last few years (see e.g. [58–60]). Like in DBI-DeLP-based applications, case-based argumentation systems tend to use relational databases to save and retrieve data, which are translated in the form of cases. As in our approach, these systems allow dynamic updates in their KBs, but LCBR systems are also able to incorporate new knowledge from the current case implicitly in their reasoning cycle. Nevertheless, it is clear that the argumentation approaches in LCBR are focused on legal reasoning, while our framework can be used also to develop other types of applications like Decision Support Systems or Recommender Systems. Thus, LCBR approaches will certainly be more effective than ours in a legal setting, but still our framework is more general and will be useful in another areas.

There have been other approaches connecting case-based reasoning with argumentation in more general settings than those of the legal one, which are connected to some extent with our proposal but from different perspectives. In [61] the authors present Hermes, a system that augments classical decision making approaches by supporting argumentative discourse among decision makers. It is fully implemented in Java and runs on the Web, thus providing relatively inexpensive access to a broad public. Advanced features of the system include enabling users to retrieve data stored in remote databases in order to further warrant their arguments, stimulating them to perform acts that best reflect their interests and intentions. In contrast, our proposal aims at providing a more general ontology for integrating argumentation and databases, whereas Hermes was intended as a tool for collaborative decision making. In [62], the authors introduce an extension of the basic mechanisms used in conventional argumentation frameworks. This extension is called Argumentation System Based on Ontologies (ASBO), and consists of a new and convenient style of attack to arguments, making explicit the argumentation process structure through an OWL-based ontology. In line with our proposal, ASBO follows an engineering-oriented approach to materialize a software architecture which allows working with argumentation in MAS. However, the focus is on persuasion dialogs among agents, whereas our approach is defined in a more generic setting oriented to integrating argumentation with relational databases, based on common basic elements in argumentation frameworks obtained from RBAS (facts, rules, etc.). In [63], a case-based approach to argumentation is presented, consisting of (1) an argumentation framework for learning agents, and (2) an individual policy for agents to generate arguments and counterarguments (including counterexamples). This proposal is mainly focused on a multi-agent learning perspective based on argumentation, and does not consider primitives for accessing and retrieving information from relational databases in the context of an argumentation framework, as done in our approach. Finally, in [64] the authors describe a negotiation model that integrates case-based reasoning, real time issues and argumentation. This model consists of two important ideas: a real-time logical negotiation protocol and a case-based negotiation model. The protocol integrates a real-time Belief-Desire-Intention (BDI) model, a temporal logic model, and communicative acts for negotiation. Contrasting with our proposal, this approach aims at characterizing an agent able to negotiate with its partners using an argumentation-based negotiation protocol, considering real-time constraints (e.g. for solving resource allocation problems). This approach also differs from ours in not taking into account the integration of relational databases as part of the characterization of the negotiation protocol.

As for future work, there are several lines of research that we plan to follow. We will describe them briefly.

One of the future goals is to enhance the presented framework with semantic information about predicates allowing to automatically obtain information about the different possible data sources. 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.

An alternate approach to cope with massive argumentation processes is to take advantage of the power of Semantic Web technologies adapted for their use in such settings. In this sense, we plan to develop methods to express relational databases in some Semantic Web processable format, and then apply solutions already developed for defeasible reasoning in such settings, e.g. [25,53,26].

Another line of research we are following is the dynamic 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, counter-examples to already known rules may arise. We plan to take advantage of such counter-examples 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 downgraded to defeasible ones, or new refined defeasible rules may be formed by analyzing the characteristics of the newly found data [65]. 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* provided by contextual queries proposed in [66].

Acknowledgements

We would like to thank to the anonymous referees for valuable comments on an earlier version of the paper.

This work has been partially supported by Universidad Nacional del Sur (UNS), Universidad Nacional de Entre Ríos (UNER), and Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), from Argentina.

Appendix A

Table 4.

Table 4
Most important acronyms used through the paper.

Acronym	Expansion
DBMS	DataBase Management System
DBI-DeLP	DataBase Integration for Defeasible Logic Programming
dbi-delp	database integrated defeasible logic program
DDH	Domain Data Holder
DDI	Domain Data Integrator
DeLP	Defeasible Logic Programming
delp	defeasible logic program
DS	Data Source
DSR	Data Source Retrieval [function]
OP	Operative Presumption
PRF	Presumption Retrieval Function
PR	Pertinence Relation
PS	Parameter Source
PTD	Predicate Translation Database
RBAS	Rule Based Argumentation Systems
SLD	Selective Linear Definite [clause resolution]
TG	Target Goal

References

[1] T. Bench-Capon, P.E. Dunne, Argumentation in artificial intelligence, Artificial Intelligence 171 (2007) 619–641.
[2] I. Rahwan, G.R. Simari, Argumentation in Artificial Intelligence, Springer, 2009.
[3] N.K. Janjua, F.K. Hussain, O.K. Hussain, Semantic information and knowledge integration through argumentative reasoning to support intelligent decision making, Information Systems Frontiers 15 (2) (2013) 167–192.
[4] 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, S. Woltran, Agreement Technologies, Law, Governance and Technology, vol. 8, Springer, New York, 2013 (Ch. 21: The Added Value of Argumentation: Examples and Challenges, pp. 357–404).
[5] G.R. Simari, A brief overview of research in argumentation systems, in: Fifth International Conference on Scalable Uncertainty Management (SUM 2011), 2011, pp. 81–95.
[6] R.P. Loui, Defeat among arguments: a system of defeasible inference, Computational Intelligence 3 (1987) 100–106.
[7] D. Nute, Defeasible reasoning: a philosophical analysis in PROLOG, Aspects of Artificial Intelligence (1988) 251–288.
[8] G.R. Simari, A Mathematical Treatment of Defeasible Reasoning and its Implementation, Ph.D. thesis, Washington University, Dep. of Comp. Science, December 1989.
[9] G.R. Simari, R.P. Loui, A mathematical treatment of defeasible reasoning and its implementation, Artificial Intelligence 53 (2–3) (1992) 125–157.
[10] H. Prakken, G. Sartor, Argument-based extended logic programming with defeasible priorities, Journal of Applied Non-Classical Logics 7 (1) (1997) 25–75.
[11] A.J. García, G.R. Simari, Defeasible logic programming an argumentative approach, Theory and Practice of Logic Programming 4 (1–2) (2004) 95–138.
[12] L. Amgoud, S. Kaci, An argumentation framework for merging conflicting knowledge bases: the prioritized case, in: Fourth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005), Springer, 2005, pp. 527–538.
[13] P.M. Dung, R.A. Kowalski, F. Toni, Dialectic proof procedures for assumption-based, admissible argumentation, Artificial Intelligence 170 (2) (2006) 114–159.
[14] H. Prakken, An abstract framework for argumentation with structured arguments, Argument and Computation 1 (2010) 93–124.
[15] J.W. Lloyd, Foundations of Logic Programmming, Springer Verlag, 1987.
[16] U. Nilsson, J. Maluszynski, Logic, Programming and Prolog, second ed., John Wiley & Sons Ltd., 1995, (A free copy could be obtained from the authors in http://www.ida.liu.se/ulfni/lpp).
[17] V. Lifschitz, Foundations of logic programs, in: Principles of Knowledge Representation, CSLI Pub., 1996, pp. 69–128.
[18] M. Gelfond, Answer sets, in: Handbook of Knowledge Representation, Foundations of Artificial Intelligence, Elsevier, 2008, pp. 285–316 (Ch. 7).
[19] M. Capobianco, C.I. Chesñevar, G.R. Simari, Argumentation and the dynamics of warranted beliefs in changing environments, Autonomous Agents and Multi-Agent Systems 11 (2) (2005) 127–151.
[20] C.A.D. Deagustini, S.E. Fulladoza Dalibón, S. Gottifredi, M.A. Falappa, C.I. Chesñevar, G.R. Simari, Supporting defeasible argumentation processes over relational databases, in: 9th International Workshop. on Argumentation in Multi-Agent Systems, ArgMAS, 2012 (in press).
[21] M.V. Martinez, A.J. García, G.R. Simari, On the use of presumptions in structured defeasible reasoning, in: Fourth International Conference on Computational Models of Argument (COMMA 2012), 2012, pp. 185–196.
[22] W.A. Carnielli, J. Marcos, Ex contradictione non sequitur quodlibet, in: II Annual Conference on Reasoning and Logic, 2001, pp. 89–109.
[23] C. Chesñevar, A. Maguitman, M. González, Argumentation in Artificial Intelligence, Springer Verlag, 2009.
[24] M. Thimm, Realizing argumentation in multi-agent systems using defeasible logic programming, in: Sixth International Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2009), 2009, pp. 175–194.
[25] S.A. Gómez, C.I. Chesñevar, G.R. Simari, Reasoning with inconsistent ontologies through argumentation, Applied Artificial Intelligence 24 (1 & 2) (2010) 102–148.
[26] I. Rahwan, Mass argumentation and the semantic web, Journal of Web Semantics 6 (1) (2008) 29–37.
[27] Website <The Internet Movie DataBase, http://www.imdb.com/>, November 2012.
[28] Website, MovieLens <http://www.movielens.org/>, November 2012.
[29] C.I. Chesñevar, A.G. Maguitman, G.R. Simari, A first approach to argument-based recommender systems based on defeasible logic programming, in: 10th International Workshop on Non-Monotonic Reasoning (NMR 2004), 2004, pp. 109–117.
[30] P. Rob, C. Coronel, Database Systems Design, fifth ed., Implementation and Management, Course Technology Press, Boston, MA, United States, 2002.
[31] P.A. Bernstein, E. Newcomer, Principles of Transaction Processing, second ed., Morgan Kaufman, 2009.
[32] L. Cecchi, P. Fillottrani, G.R. Simari, On the complexity of delp through game semantics, in: 11th International Workshop on Nonmonotonic Reasoning (NMR 2006), 2006, pp. 386–394.
[33] J.B. Grimson, Integrating knowledge-based systems and databases, Clinica Chimica Acta 222 (1993) 101–115.
[34] E. Laenens, D. Vermeir, Advanced knowledge-base environments for large database systems, Knowledge-Based Systems 3 (4) (1990) 215–220.
[35] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From data mining to knowledge discovery in databases, AI Magazine 17 (3) (1996) 37.
[36] L.A. Kurgan, P. Musilek, A survey of knowledge discovery and data mining process models, The Knowledge Engineering Review 21 (2006) 1–24.
[37] D. Michie, Machine learning: principles and techniques: Richard forsyth, Knowledge Based Systems 3 (1) (1990) 59.
[38] M. Williams, G. Chen, D. Ferbrache, P. Massey, S. Salvini, H. Taylor, K. Wong, Prolog and deductive databases, Knowledge-Based Systems 1 (3) (1988) 188–192.
[39] R. Ramakrishnan, J.D. Ullman, A survey of research on deductive database systems, Journal of Logic Programming 23 (1993) 125–149.
[40] M. Jarke, Y. Vassiliou, Coupling expert systems with database management systems, Information Systems Working Papers Series, 1983.
[41] T. Imanaka, K. Uehara, J. Toyoda, Integration of prolog and databases in both evaluational and non-evaluational approaches, Knowledge-Based Systems 2 (4) (1989) 203–209.
[42] C.L. Chang, A. Walker, PROSQL: a Prolog programming interface with SQL/DS, in: First International Workshop on Expert Database Systems, 1986, pp. 233–246.
[43] S. Ceri, G. Gottlob, G. Wiederhold, Efficient database access from prolog, IEEE Transactions on Software Engineering 15 (2) (1989) 153–164.
[44] R.J. Lucas, G.A. Le Vine, Prolog and Databases: Implementations and New Directions, Halsted Press, New York, NY, USA, 1988 (Ch. A Prolog-relational database interface, pp. 67–80).
[45] S. Ceri, G. Gottlob, L. Tanca, What you always wanted to know about datalog (and never dared to ask), IEEE Transactions on Knowledge Data Engineering 1 (1) (1989) 146–166.
[46] R. Girle, D. Hitchcock, P. Mcburney, B. Verheij, Argumentation Machines. New Frontiers in Argument and Computation, Kluwer Academic Publishers, 2003 (Ch. Decision Support For Practical Reasoning: a theoretical and computational perspective, pp. 55–84).
[47] N.D. Rotstein, A.J. García, G.R. Simari, Reasoning from desires to intentions: a dialectical framework, in: 22nd AAAI Conference on Artificial Intelligence, Canada, 2007, pp. 136–141.
[48] C.E. Briguez, M.C. Budán, C.A.D. Deagustini, A.G. Maguitman, M. Capobianco, G.R. Simari, Towards an argument-based music recommender system, in: Fourth International Conference on Computational Models of Argument (COMMA 2012), 2012, pp. 83–90.
[49] M. Wardeh, T.J.M. Bench-Capon, F. Coenen, Padua: a protocol for argumentation dialogue using association rules, Artificial Intelligence Law 17 (3) (2009) 183–215.
[50] M. Wardeh, F. Coenen, T.J.M. Bench-Capon, Pisa: a framework for multiagent classification using argumentation, Data Knowledge Engineering 75 (2012) 34–57.
[51] S. Ontañón, E. Plaza, Multiagent inductive learning: an argumentation-based approach, in: 27th International Conference on Machine Learning (ICML 2010), 2010, pp. 839–846.
[52] S. Ontañón, P. Dellunde, L. Godo, E. Plaza, A defeasible reasoning model of inductive concept learning from examples and communication, Artificial Intelligence 193 (2012) 129–148.
[53] N. Bassiliades, G. Antoniou, I.P. Vlahavas, A defeasible logic reasoner for the semantic web, International Journal of Semantic Web Information System 2 (1) (2006) 1–41.
[54] C. Chesñevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, S. Willmott, Towards an argument interchange format, Knowledge Engineering Review 21 (4) (2006) 293–316.
[55] S. Brüninghaus, K.D. Ashley, Predicting outcomes of case-based legal arguments, in: Ninth International Conference on Artificial Intelligence and Law (ICAIL 2003), 2003, pp. 233–242.
[56] V. Aleven, K.D. Ashley, Teaching case-based argumentation through a model and examples: empirical evaluation of an intelligent learning environment, Artificial Intelligence in Education 39 (1997) 87–94.
[57] E.L. Rissland, K.D. Ashley, L.K. Branting, Case-based reasoning and law, The Knowledge Engineering Review 20 (2005) 293–298.
[58] H. Prakken, A. Wyner, T. Bench-Capon, K. Atkinson, A formalization of argumentation schemes for legal case-based reasoning in aspic+, Journal of Logic and Computation (2013). http://dx.doi.org/10.1016/j.knosys.2013.07.010.
[59] A.Z. Wyner, T.J.M. Bench-Capon, K. Atkinson, Towards formalising argumentation about legal cases, in: Thirteenth International Conference on Artificial Intelligence and Law (ICAIL 2011), 2011, pp. 1–10.
[60] A.Z. Wyner, T.J.M. Bench-Capon, Argument schemes for legal case-based reasoning, in: 20th Anniversary International Conference on Legal Knowledge and Information Systems (JURIX 2007), 2007, pp. 139–149.
[61] N.I. Karacapilidis, D. Papadias, Computer supported argumentation and collaborative decision making: the hermes system, Information Systems 26 (4) (2001) 259–277.
[62] A. Muñoz, J.A. Botía, Asbo: Argumentation system based on ontologies, in: 12th International Workshop Cooperative Information Agents (CIA 2008), 2008, pp. 191–205.
[63] S. Ontañón, E. Plaza, Arguments and counterexamples in case-based joint deliberation, in: Third International Workshop Argumentation in Multi-Agent Systems (ArgMAS 2006), 2006, pp. 36–53.

[64] L.-K. Soh, C. Tsatsoulis, A real-time negotiation model and a multi-agent sensor network implementation, Autonomous Agents and Multi-Agent Systems 11 (3) (2005) 215–271.
[65] M.A. Falappa, G. Kern-Isberner, G.R. Simari, Explanations, belief revision and defeasible reasoning, Artificial Intelligence 141 (1/2) (2002) 1–28.
[66] S. Gottifredi, A.J. García, G.R. Simari, Query-based argumentation in agent programming, in: 12th Ibero-American Conference on AI (IBERAMIA 2010), 2010, pp. 284–295.