DBI-DeLP: a Framework for Defeasible Argumentation over Databases

C. A. D. Deagustini, S. E. Fulladoza Dalibón, S. Gottifredi, and G. R. Simari

[1] Laboratorio de Investigación y Desarrollo en Inteligencia Artificial,
Departamento de Ciencias de la computación e Ingeniería,
Universidad Nacional del Sur - Av. Alem 1253, (8000) Bahía Blanca, Buenos Aires
[2] Área Agentes y Sistemas Inteligentes,
Universidad Nacional de Entre Ríos - Av. Mons. Tavella 1424, (3200) Concordia,
Entre Ríos
{cadd,sef,sg,grs}@cs.uns.edu.ar

Abstract. Nowadays Argumentation Systems in general, and DeLP in particular, build arguments based on the context of a single and fixed logical program. This leads to a practical limitation regarding the volume of data in which the argumentation is supported, because integration of constantly updated external data only can be made by the "hard-coding" of facts (i.e., the explicit codification of facts in the program), which is inneficient for massive data.
This paper introduces Database Integration for Defeasible Logic Programming (DBI-DeLP), a framework that integrates Defeasible Argumentation with Databases that may be updated by other external applications, allowing the execution of argumentation processes based on masive external sources of data.

Keywords: Defeasible Argumentation, Databases.

1 Introduction

In the last decade argumentation has emerged as a sophisticated mechanism in formalization of common sense reasoning finding application in diferent fields of articial intelligence like legal systems, multi-agents negotiation and Decision Support Systems among others, as can be seen in [2, 11, 4], and they have proven its utility in those fields. Intuitivelly, in an argumentation system, an argument is a piece of tentative information that supports a claim. The acceptance of this claim will depend of a dialectical analysis (formalized through a proof procedure) of the arguments in favor and against the claim and its argument[11]. Among Argumentation Systems there are a particular kind called Rule Based Argumentation Systems (RBAS) [10, 6, 1, 8]. In these kind of AS, arguments are built from a specific knowledge base of inference rules, usually known as program. Therefore an argument in these systems will be a set of rules from which the claim of the argument can be inferred. In particular, these AS are specially interesting in Artificial Intelligence (AI), because they allow common sense reasoning and automatic argument building.

However, and besides its utility, Rule Based Argumentation Systems have some drawbacks. Generally, argument are built only from *local* information, making them more subjectives, i.e., more affected by the agent's particular vision of the domain. Also, updating of knowledge in a continuous way is inefficient due to the need of explicit codification of rules and facts in the program, and there are no tools for the efficient incorporation of external applications's data.

Here we present *Database Integration for Defeasible Logic Programming (DBI-DeLP)*, a framework that enables common sense reasoning over data stored in Databases. For this, DBI-DeLP uses DeLP[8] to handle the argumentation process, feeding it with the information from available domain data sources.

Like DeLP, in DBI-DeLP two kinds of information are considered: those that are regarded as strict knowledge (which must preserve internal coherence, i.e., they can't be contradictory) for example actor ←sean_penn; and those that are weak or tentative (which can be used if nothing could be posed against it), for example good_movie(Movie) $-<$ genre(Movie, action), performs_in(Movie, schwarzenegger). This last one is interpreted as "an action movie in which Arnold Schwarzenegger performs in is usually a good movie" Clearly information obtained by using defeasible rules is tentative and does not need to be consistent with other obtained this way, and using this kind of rules both a literal and its complement can be derived, enabling the inclusion in a knowledge base of contradictory information.

Since information stored in a database is potentially contradictory (both in the sense of opposite data or data that can lead to complementary conclusions), and given that we don't want to restrict the content in this sense, database obtained knowledge is represented as "weak facts" called *Presumptions*[9] denoted *Head*$-<$ and interpreted as "there are reasons to believe in Head".

The paper is organized as follows: in **Section 2** we review DeLP, the formalism that supports DBI-DeLP; in **Section 3** a possible structure that allows the realization of argumentation processes over information stored in databases is shown, and finally in **Section 4** conclusions reached are given, and future lines of work are identified.

2 Background

In this section we give a brief summary of Defeasible Logic Programming (DeLP). DeLP is a formalism that combines results of Logic Programming and Defeasible Argumentation. DeLP provides the possibility of representing information in the form of rules in a declarative manner, and a defeasible argumentation inference mechanism for warranting the entailed conclusions. These rules are the key element for introducing defeasibility and they will be used to represent a relation between pieces of knowledge that could be defeated after all things are considered. Using these rules, common sense reasoning is defeasible in a way that is not explicitly programmed.

A Defeasible Logic Program (or de.l.p. for short) is a pair (Π, Δ) where Π is a set of strict rules and facts, and Δ is a set of defeasible rules. In a

de.l.p. knowledge can be represented using strict rules, facts and defeasible rules. Facts are ground literals representing atomic information or the negation of atomic information using strong negation "$\sim$" (e. g. a, or $\sim$ a). Strict rules are denoted $L_0 \leftarrow L_1, ..., L_n$, and represents information that can not be refused, *i.e.* if Body can be proven then Head is granted. Defeasible Rules (d-rules) are denoted $L_0 -\!< L_1, ..., L_n$. A d-rule represents tentative information that may be used if nothing could be posed against it. A d-rule $Head -\!< Body$ expresses that "reasons to believe in the antecedent Body give reasons to believe in the consequent Head".

Definition 1. *(Facts, Strict rules and Defeasible rules)*
Given a literal L_0, i.e. a ground atom or a negated ground atom, and a finite not empty set of literals Body in the form $L_1, \ldots, L_n$:
- *A fact is a literal L_0 denoted "$L_0 \leftarrow$".*
- *A strict rule is an ordered pair "$L_0 \leftarrow Body$".*
- *A defeasible rule is an ordered pair "$L_0 -\!< Body$".*

A defeasible rule with an empty body is called a presumption [9]. Presumptions are assumed to be true if nothing could be posed against them. In [8] an extension to DeLP that includes presumptions is presented, where an extended de.l.p. is a set of facts, strict rules, defeasible rules and presumptions.

Definition 2. *(Presumption) A presumption is a literal. Given a literal L_0, a presumption is denoted "$L_0 -\!<$".*

From a DeLP program it will be posible to infer tentative information. These inferences are called defeasible derivations, and are computed by backward chaining aplying the usual SLD inference procedure used in logic programming.

Definition 3. *(Defeasible derivation) Let $\mathcal{P} = (\Pi, \Delta)$ be a* de.l.p *and L a ground literal. A defeasible derivation of L from $\mathcal{P}$, denoted $\mathcal{P} \vdash L$, consists of a finite sequence $L_1, L_2, \ldots, L_n = L$ of ground literals, and each literal L_i is in the sequence because:*
(a) L_i is a fact or a presumption,
(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 strong negation can appear in facts or in the head of defeasible rules, observe that from a de.l.p. contradictory literals could be derived. However, the set Π (used to represent non-defeasible information) must be non-contradictory, i. e. no pair of contradictory literals can be derived from Π. Given a literal L, $\bar{L}$ represents the complement with respect to strong negation.

In DeLP when contradictory literals are derived, a dialectical process is used for deciding which literals are warranted. A literal L is warranted if there exists a non-defeated argument A for L. An argument for a literal L, denoted $\langle A, L_i \rangle$, is a minimal non-contradictory set of d-rules $A \subseteq \Delta$, that allows to derive L.

To establish if $\langle A, L_i \rangle$ is a non-defeated argument, defeaters for $\langle A, L_i \rangle$ are considered. Counterarguments of $\langle A, L_i \rangle$ are those arguments that disagree (are in contradiction) at some point with $\langle A, L_i \rangle$. A counter-argument is a defeater of argument A if it is preferred to $\langle A, L_i \rangle$ by some argument comparison criterion.

In this work we will not focus on the dialectical proof procedure used by DeLP in order to determine which arguments prevail. More detailed information regarding how the warrant procedure is performed can be found in [8].

3 A Defeasible Argumentation over Databases Framework

DeLP provides a suitable framework for building real-world applications that deal with incomplete and potentially contradictory information, but this kind of applications usually generates massive amounts of data which are generally stored in databases. Clearly, including this information into a DeLP program by means of hard-coding of rules and facts is inefficient. Nevertheless, this new data should be considered when new arguments and counter-arguments are builded, as part of the system's knowledge evolution. As an example, consider the legal enviroment. New legal precedents are established each time a case resolution is given, and they obviously have to be taken into consideration by the DeLP-builded application in subsequent argument processes. Besides, being databases the most popular digital storage system this days, digital records of cases will certainly be supported this way, and probably there will be public available legal databases storing potencially argument supporting data being updated by other systems. Considering all things mentioned, to develop a framework that allows the use of database stored knowledge by a Defeasible Argumentation language DeLP will grant DeLP builded applications the possibility of give much more precise and fair answers to queries they receive. In this section we introduce DBI-DeLP, a framework that integrates DeLP with DataBases.

First we define elements included in DBI-DeLP formalism, next we identify the elements that integrates the framework and describe their purpose, and then we show the interaction between them.

Knowledge about a certain domain by different entities can be contradictory. Since a DBI-DeLP based system may use information provided by several entities through their databases, the use of facts to represent them is not possible because it can lead to inconsistencies in the set of strict knowledge Π. Due to that, in DBI-DeLP we introduce *Operative Presumptions*, which are presumptions used to represent such knowledge.

Definition 4. *(Operative Presumption) An operative presumption is a literal, i.e. a negated or not negated ground atom, denoted "Head–<" such that Head represents information stored in a relational database used by the system.*

A DBI-DeLP program is a set of strict rules, facts, defeasible rules and operative presumptions, where the set Π of strict rules and facts and the set Δ of defeasible rules are fixed in the program's logic; while the set Σ of operative

presumptions is changed when databases it represents are modified. Σ is the set of all potencial operative presumptions that can be obtained from information mantained in all the databases the system uses. We will show in section 3.2 that the operative presumptions are dynamically builded only when needed for each query the DBI-DeLP Server receives by means of a *Presumption Retrieval* function, and discarded later when the query has been solved.

Definition 5. *(DBI-DeLP Program) A DataBase-Integrated Defeasible Logic Program $\mathcal{P}$ is a triplet(Π, Δ, Σ) where Π is a set of facts and strict rules, Δ is a set of defeasible rules, and Σ is a set of Operative Presumptions.*

3.1 Components of the Framework

Enabling Database records to be used by the inference process in DeLP involve several aspects: first, accesible databases has to be identified, and access information has to be mantained; also easy and fast addition of databases if they became accesible is desirable, and finally both the DeLP inference mechanism and schemas of databases it uses must remain unchanged so compatibility with external systems is mantained, so a "traduction layer" between them is needed. In order to do that, DBI-DeLP components that carried out different jobs are defined. There are three major components in the Framework: *DBI-DeLP Server*, which contains the DeLP core and is used to receive queries, solve them and give answers; *Domain Data Holder*, which is a set of databases that keeps data about the domain, and finally *Domain Data Integrator* which is the component that bridges DeLP and Databases, that is, it translates a DeLP query into a SQL query and turns the SQL answer into a DeLP specification. Each component of the framework will be described next.

DBI-DeLP Server The DBI-DeLP Server component is in charge of getting DeLP ground queries, build arguments and its counter-arguments based on the knowledge it has, and giving answers and explanations of how they were builded. It comprises two different parts, the execution of the inference mechanism, and the knowledge that supports the results of the inference. This two aspects are included in the DBI-DeLP server by two modules: DeLP Core and Domain Logic.

DeLP Core The argumentation process is carried out by the DeLP core. Basically it receives a query like *good_movie(commando)?* from a client and tries to build arguments for and against it, and give an answer that can be **Yes** if an argument in favor is warranted, **No** if an argument against it is warranted, **Undecided** if neither arguments for or against it can be warranted, and **Unknown** if the query includes literals that are not in the program´s language.

Domain Logic Domain Logic is the knowledge of the Domain the system has. It is expressed as a DBI-DeLP program. For example a movie Argument-based Recommender System[3] using DBI-DeLP could have strict rules like child_restricted(Movie) $\leftarrow$ has_violence(Movie) and defeasible rules like has_violence(Movie) $-\!\prec$ director(Movie, quentin_tarantino), and dinamically add operative presumptions like film_genre(pirates_of_the_caribbean, comedy) and

film_genre(pirates_of_the_caribbean, action) if different categorizations for the film Pirates of the Caribbean are founded in the domain data.

Domain Data Holder (DDH) Domain Data Holder is a masive, potencially contradictory set of domain related data which is used for founding grounds in the argument building process. In the current version of the framework, the data is stored in independent relational databases which are accesed via an Open DataBase Connectivity (ODBC) driver, allowing databases to be in any DataBase Management System (DBMS) that has its ODBC driver implemented, like MySQL, SQLite or dBASE. Every database in the DDH has to be set up with its own ODBC connection before it can be used by the DBI-DeLP server. There isn't a theoretical limit in the number of databases included in the DDH, and the addition or remove of a database has no effect on the others (but obviosly the knowledge is altered so if a previous query is launched again the answer obtained may vary). Also there aren't restrictions about how tables and fields should be named, or how the database schema should be, but configuration for each database is needed so the server knows what tables and fields to include in the SQL query it sends to the DBMS. Nonetheless configuration is as simple as adding rows to tables in a database that keeps information about relations between predicates and databases in the DDH.

Domain Data Integrator (DDI) As said before, we will not change the representational structure of DeLP and the databases it will use. Therefore, in order to enable the communication between them, we should establish an intermediate layer. Integration between the DBI-DeLP Server and the DDH is made by the final component in the framework architecture: the Domain Data Integrator (DDI), which is responsible for transforming DeLP queries into SQL queries, and formatting resulting Datasets into presumptions so the data can be used by DeLP in the argumentation process. Information needed by the DDI to performs traductions is provided by the *Predicate Traduction Database*.

Predicate Traduction DataBase As stated before, this database mantains information about relations between predicates and databases. In this version of the framework it has four tables:

- Predicates table: has information about correspondence between predicate's functors and data sources.
- Parameters table: mantains the equivalence between a predicate's parameter and a pair (table, field).
- RelatedTables table: keeps information about the tables that take part in the SQL JOINS needed to obtain information about a particular predicate.
- ForeignKeys table: mantains a list of the pairs (primaryKey, foreignKey) on which the SQL JOINS have to be made.

3.2 Query Solving Process in DBI-DeLP

In this section we discuss the interaction between the components that integrates the framework in the query solving process. Figure 1 show how this is done. The process is executed each time DBI-DeLP Server receives a query.

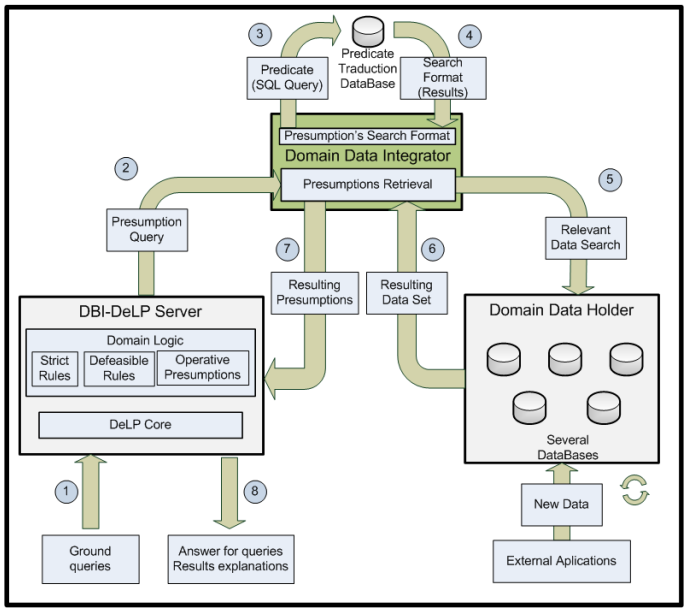

Fig. 1. The DBI-DeLP argumentation process

When the defeasible derivation procedure in DBI-DeLP is looking support for a literal, besides the search for strict rules, facts and defeasible rules as in DeLP, a search for presumptions (using condition (a) of **Definition 3**) is launched, in order to retrieve from the databases, if exists, some information offering support to the literal. This involves determining what databases (and which tables and fields) are expected to have useful data for the literal, getting that data if it exists, and make them available to the DeLP core which build answers to the query based in this information (along with the rest of the Domain Logic), and finally give the answers to the client that has realized the query. Therefore, for each strict rule Head $\leftarrow$ Body or defeasible rule Head $-\!\!<$ Body, where the inference procedure is trying to prove Head (in order to resolve a received query), there will be a call to the *Presumption Retrieval* function for every literal L_i in Body, which represents the function objectives.

Definition 6. *(Function Objective) Given a strict rule* $L_0 \leftarrow L_1, \ldots, L_n$*, or a defeasible rule* $L_0 -\!\!< L_1, \ldots, L_n$*, where* L_0 *is the literal the inference procedure is trying to prove, we call Function Objective to each literal* L_i *(*$1 \leq i \leq n$*) in the form* $func(p_1, \ldots, p_n)$*, where* $func$ *is the predicate's functor and* $p_1, \ldots, p_n$ *is a list of the predicate's parameters.*

The *Presumption Retrieval* function makes use of the follow functions in order to setup the SQL Queries DBI-DeLP needs to execute and format the SQL results so the DeLP Core can use them. The *Obtain Instanciated Parameters function* receives a list of paremeters from an objective literal and returns those that

are ground. Then the *Obtain Instanciated Fields function* takes a list of fields and a list of parameters and returns those fields corresponding to instanciated parameters. Finally, the *Generate Operative Presumption function* receives a functor's name and a list of values and returns a predicate where functor's name is the predicate's functor and the list of values is the predicate's parameters.

Next we present, by means of an algorithm, the process used by DBI-DeLP to obtain Operative Presumptions for a Function Objective.

Algorithm 1 Presumption Retrieval

1: **function** PRESUMPTIONRETRIEVAL(funcObjective L_i):operativePresumptionsList
2: Decompose L_i into its Functor func and a list of parameters $p_1, \ldots, p_n$
3: ***instanciatedParameters*** ← obtainInstanciatedParameters($p_1, \ldots, p_n$)
4: Execute a SQL Query in the form "SELECT DSN, User, Pass FROM predicates WHERE name = func"
5: **for** *each DSN obtained* **do**
6: ***fieldsToRetrieve*** ← Execute a SQL Query in the form "SELECT Table, Field FROM parameters WHERE parameters_dsn_id = predicates_dsn_id"
7: ***whereFields*** ← obtainInstanciatedFields(fieldsToRetrieve, $p_1, \ldots, p_n$)
8: Execute a SQL Query in the form "SELECT Table, Field FROM arguments WHERE arguments_dsn_id = predicates_dsn_id"
9: ***joinTables*** ← Execute a SQL Query in the form "SELECT Table FROM relatedTables WHERE relatedTables_dsn_id = predicates_dsn_id"
10: ***joiningFields*** ← Execute a SQL Query in the form "SELECT Table, Field FROM foreignKeys WHERE foreignKeys_dsn_id = predicates_dsn_id"
11: results ← Execute a SQL Query in the form "SELECT ***fieldsToRetrieve*** FROM ***joinTables*** ON ***joiningFields*** WHERE ***whereFields*** = ***instanciatedParameters***"
12: **for** *each result* res_i *obtained* **do**
13: operativePresumptionList[i] ← generateOperativePresumption($func$, res_i)
14: **end for**
15: **end for**
16: **return** *operativePresumptionsList*

Finally, we show an example of the complete query solving process in DBI-DeLP:

1. DBI-DeLP server receives a query like *good_movie(demolition_man)?*.
2. DeLP core starts the search of a rule that has good_movie(X) as its head and finds *good_movie(Movie)–<performs_in(Movie, Actor), famous(Actor).*, so the server tries to find proofs for performs_in(demolition_man, Actor). The proof search occurs in all the Domain Logic, but for this paper purpose we will only show how presumptions are obtained.
3. In order to do that, DDI takes performs_in(demolition_man, Actor) and decompose it in two parts: its functor performs_in, and a list of its parameters, in this case [demolition_man, Actor]. Then the DDI looks in the Predicate Traduction DataBase for information about performs_in, searching for the ODBC Data Source Name (DSN), tables and fields related to the predicate.

4. DBMS that manages the Predicate Traduction DataBase answers the queries the DDI sent. That way DDI knows which tables to look for information, and how the fields are named. For the sake of simplicity, let's assume that we have in our DDH a database with a table named film_actor that has one field called film and another called actor_name. Observe how the name inconsistency between the predicate called performs_in and the table called film_actor is solved by means of a syntactic-level traduction.
5. Now that knows where to look for information, DDI executes SQL queries using that information. In this example, it will execute "SELECT film, actor_name FROM film_actor WHERE film = demolition_man" to the database the DSN points to. Notice that only grounded arguments are used for WHERE conditions by the DDI.
6. The DBMS answers the query with the actors that performs in the movie Demolition Man. The results are obtained in a list where each element has a row(demolition_man, actor's name) format.
7. DDI takes that results and format them as Operative Presumptions. For example, if we have two results row(demolition_man, sylvester_stallone) and row(demolition_man, wesley_snipes), they are transformed to *performs_in(demolition_man, sylvester_stallone)* $-\!\prec$ *true.* and *performs_in(demolition_man,wesley_snipes)*$-\!\prec$*true.*, according to DeLP's syntax. Then all the results obtained are sent to the DeLP core as a list of presumptions. DeLP core unifies the second argument on each result with the argument in famous(Actor), so for example it now will search proofs for famous(sylvester_stallone), starting the process all over again.
8. The argument and counter-argument building process is carried out, and then answers and explanations are sent to the client. For example, if famous(sylvester_stallone) can be proved, good_movie(demolition_man) succeed, and if there are no counter-arguments the answer is **YES**.

4 Conclusions and Related Work

We have shown how DeLP can be combined with DataBase technologies in order to achieve argumentation over a large amount of data. This approach is more efficient than explicit codification of facts because other systems may "give" data to ours without the need of complex interfaces. This can lead to definitions of new architectures for Argument-based Recommender Systems[3], as well as Desicion Support Systems (DSS). Formalism was introduced, an architecture for a framework that allows argumentation over databases has been presented, and the process of obtaining data from databases and using them in argument building was shown. As for future work, there are several lines identified: Semantic-level traduction between predicates and database schemas research is desirable, efficiency test with several huge databases needs to be performed, reasoning over databases schemas, mechanisms for resolving non-ground queries have to be developed, and new rules's learning based on obtained data.

To the best of authors's knowledge, there isn't work done regarding to the integration of relational databases technologies with defeasible argumentation

systems. However, there is research related to the use of non-monotonic reasoning to resolve inconsistencies in databases by means of *database repair* [7, 12]. Altough this approach can resolve inconsistencies in databases allowing reasoning on data stored in them, database repair is conducted by the explicit addition, modification or suppression of tuples in databases. Instead we have used a conflict resolution strategy based in defeasible argumentation, therefore, in our approach there no modification of information stored in the databases.

Another approach that uses databases as the basis of a reasoning process is presented in [5]. The aim of the reasoning mechanism is to address the diferent conflicts that may arise when merging several databases. Nevertheless, databases used in this work are deductive databases, *i.e.* databases that are made of an extensional part (a set of positive or negative ground literals) and an intensional part (a set of first order function-free clauses). Deductive databases have some known drawbacks that are not present in DBI-DeLP, like the need to define criteria for using a law included in the database as a deduction rule or a coherence rule. In DBI-DeLP this does not happen because rules are only used to build arguments. Also other known drawback is that in deductive databases there is the posibility of infinite loops in the deduction process. This is also avoided in DBI-DeLP with constraints over the argumentation lines [8].

References

1. Amgoud, L., Kaci, S.: An argumentation framework for merging conflicting knowledge bases: The prioritized case. In: In Proc. of the ECSQARU-2005 Conf., LNAI 3571. pp. 527–538. Springer (2005)
2. Bench-Capon, T.J.M., Dunne, P.E.: Argumentation in artificial intelligence. Artif. Intell. 171, 619–641 (July 2007)
3. Chesñevar, C.I., Maguitman, A.G., Simari, G.R.: A first approach to argument-based recommender systems based on defeasible logic programming. In: In Proc. 10th Intl. Workshop on Non-Monotonic Reasoning. pp. 109–117 (2004)
4. Chesñevar, C.I., Maguitman, A.G., Simari, G.R.: Argument-based user support systems using defeasible logic programming. In: AIAI. pp. 61–69 (2006)
5. Cholvy, L., Garion, C.: Answering queries addressed to several databases according to a majority merging approach. J. Intell. Inf. Syst. 22, 175–201 (March 2004)
6. Dung, P.M., Kowalski, R.A., Toni, F.: Dialectic proof procedures for assumption-based, admissible argumentation (2005)
7. Eiter, T., Fink, M., Greco, G., Lembo, D.: Repair localization for query answering from inconsistent databases. ACM Trans. Database Syst. 33(2) (2008)
8. García, A.J., Simari, G.R.: Defeasible logic programming an argumentative approach. TPLP pp. 95–138 (2004)
9. Nute, D.: Defeasible reasoning: a philosophical analysis in prolog. In: Fetzer, J. (ed.) Aspects of Artificial Intelligence. pp. 251–288. Kluwer AP (1988)
10. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. Journal of Applied Non-Classical Logics 7(1) (1997)
11. Rahwan, I., Simari, G.R.: Argumentation in Artificial Intelligence. Springer (2009)
12. Santos, E., Martins, J.a.P.a., Galhardas, H.: An argumentation-based approach to database repair. In: 19th European Conference on Artificial Intelligence (ECAI). pp. 125–130 (2010)