Argument and Computation, 2014
Vol. 5, No. 1, 63–88, http://dx.doi.org/10.1080/19462166.2013.869767

Defeasible logic programming: DeLP-servers, contextual queries, and explanations for answers

Alejandro J. García[a,b] and Guillermo R. Simari[a*]

[a]*Artificial Intelligence Research and Development Laboratory, Department of Computer Science and Engineering, Universidad Nacional del Sur, Alem 1253, Bahía Blanca 8000, Argentina;* [b]*Consejo Nacional de Investigaciones Científicas y Técnicas, Buenos Aires, Argentina*

(*Received 1 November 2013; final version received 19 November 2013*)

Argumentation represents a way of reasoning over a knowledge base containing possibly incomplete and/or inconsistent information, to obtain useful conclusions. As a reasoning mechanism, the way an argumentation reasoning engine reaches these conclusions resembles the cognitive process that humans follow to analyse their beliefs; thus, unlike other computationally reasoning systems, argumentation offers an intellectually friendly alternative to other defeasible reasoning systems. Logic Programming is a computational paradigm that has produced computationally attractive systems with remarkable success in many applications. Merging ideas from both areas, Defeasible Logic Programming offers a computational reasoning system that uses an argumentation engine to obtain answers from a knowledge base represented using a logic programming language extended with defeasible rules. This combination of ideas brings about a computationally effective system together with a human-like reasoning model facilitating its use in applications.

Keywords: argumentation frameworks; defeasible reasoning; defeasible logic programming

1. Introduction

The representation of knowledge and its exploitation in solving problems is one of the most important areas of Artificial Intelligence. In this fertile field of research, described as Knowledge Representation and Reasoning, different systems have been proposed during the past decades, offering interesting and highly valuable alternatives such as McCarthy's Circumscription (McCarthy, 1980), Reiter's Default Logic (Reiter, 1980), McDermott and Doyle's Nonmonotonic Logic (McDermott & Doyle, 1980), Moore's Autoepistemic Logic (Moore, 1984), and the work of Pollock on defeasible reasoning (Pollock, 1987, 1996) and Loui on argumentation (Loui, 1987), started lines of research that lead to many proposals to address the problem of having a computational approach to commonsense reasoning.

The research started in the 1980s led to a number of proposals on computationally oriented systems that consider the theoretical and practical underpinnings of argumentation as an effective reasoning process. Particularly, in the argumentation community we can mention the work of Lin and Shoham (1989), Nute (1987, 1988), Simari (1989) and Simari and Loui (1992) considering defeasible reasoning and argumentation, Dung (1993, 1995) on abstract argumentation frameworks, Bondarenko, Toni, and Kowalski (1993) and Bondarenko, Dung, Kowalski, and Toni (1997) leading to Assumption-Based Argumentation, Prakken (2010) and Modgil and Prakken (2013) on the development of ASPIC+, Besnard and Hunter introduced in Besnard and Hunter (2001, 2008) an argumentation system based on classical logic. Several works on the research on

*Corresponding author. Emails: grs@cs.uns.edu.ar; grsimari@gmail.com argumentation systems have been produced, see Chesñevar, Maguitman, and Loui (2000), Prakken and Vreeswijk (2002), Bench-Capon and Dunne (2007), Besnard and Hunter (2008), and Rahwan and Simari (2009).

From the body of work on Nonmonotonic and Defeasible Reasoning, Logic Programming has emerged as one of the more attractive choices for its theoretical soundness and effective implementations. Thus, systems based on this paradigm have received increasing attention from industry for the construction of intelligent systems. The introduction of the elements of defeasible reasoning in logic programming was a natural choice to address the problem of inconsistency.

Defeasible Logic Programming, or DeLP for short, provides a computational reasoning system that uses an argumentation engine to obtain answers from a knowledge base represented using a logic programming language extended with defeasible rules that stem from the work reported in Simari (1989) and Simari and Loui (1992). The language of the DeLP system contains classical negation and offers the possibility of using default negation (see Baral & Gelfond, 1994), sharing the declarative capability of logic programming for representing knowledge, and adding the possibility of representing weak information in the form of *defeasible rules*. The argumentation-based inference mechanism has the ability of considering reasons for and against potential conclusions and deciding which are the ones that can be obtained (warranted) from the knowledge base (see García, 2000, García & Simari, 2004).

The DeLP reasoning engine forms the core of a Defeasible Logic Programming Server, or DeLP-Server for short (García, Rotstein, Tucat, & Simari, 2007). A DeLP-Server is conceived as the support for the implementation of argumentative reasoning services that work over knowledge bases represented as defeasible logic programs. These servers were developed to provide a reasoning service as part of a knowledge-based infrastructure in multi-agent systems (MAS). An inherent characteristic of MASs is that they constitute a distributed environment, where hosts for services and agents could be in different locations that could even be physically apart; thus, client agents that operate in an MAS, can remotely access DeLP-Servers that support different reasoning services that exploit particular knowledge bases.

In addition to answering queries from a knowledge base,[1] a DeLP-Server offers the additional capability of complementing these queries with an additional fragment of a defeasible logic program. This program fragment will act as a private context for the query, allowing a DeLP-server to respond to that contextualised query using not only the knowledge stored in the server but also temporarily modifying it with this private context in several specific ways. We will discuss the subject in one of the sections below.

Another important and desirable feature in any knowledge-based system is that of offering explanations that will improve the understanding of an answer returned after a query. In the framework discussed here, explanations aim to transfer the understanding of how, from a given argumentation framework, the status of a particular argument was obtained; thus, an explanation will consist of a structure that reflects the analysis carried out to obtain such status, and it will contain those arguments and counterarguments that were considered in the analysis leading to its warrant status. That is, the warrant status of a claim will be explained by presenting the whole set of dialectical trees that was generated to decide the warrant status of the arguments related to the claim. From the point of view of the receiver, this explanation will make available all the arguments considered, their statuses (i.e. defeated/undefeated), with all their interactions explicitly shown, showing to the receiver of the explanation all the elements at stake in the dialectical analysis that supports the answer.

This paper has the aim of presenting in a progressive manner the elements involved in a knowledge-based system where its reasoning mechanism uses an argumentative approach to epistemic reasoning. We will begin by introducing the essential constituents of DeLP, then we will proceed to describe how a DeLP-Server works, and we will introduce the explanation facility designed with argumentative reasoning in mind. To conclude this work, we will succinctly describe other developments based on DeLP.

2. Knowledge representation

DeLP is a formalisation of defeasible reasoning in which results of Logic Programming and Argumentation are combined. DeLP has the declarative capability of representing knowledge in a language that extends the language of logic programming with the possibility of representing weak information in the form of *defeasible rules*, and an argumentation-based inference mechanism for warranting conclusions. In what follows, we will use concepts of Logic Programming that will be brought in as necessary in the presentation.

This section will introduce a description of DeLP's features for knowledge representation and then, in the following section, the details concerning its inference mechanism will be explained. Although the work leading to the formalisation of DeLP began in the early 1990s Simari, Chesñevar, García (1994a, 1994b) as an evolution of the work in Simari and Loui (1992), its complete formalisation was completed in García (2000) and finally published in García and Simari (2004). Further developments, that will be described below, can be found in the following related material in García et al. (2007), Tucat, Garcia, and Simari (2009), Martínez, García, and Simari (2012), and García, Chesñevar, Rotstein, and Simari (2013).

Our knowledge representation language will be determined by a set of *atoms*. Atoms can be preceded by the *strong negation* symbol '$\sim$'. Atoms that are not preceded by strong negation will also be called *positive literals* and atoms preceded by strong negation will be called *negative literals*, the term *literal*, or sometimes *objective literal*, will refer to either one. A pair of literals involving a positive and a negative literal over the same atom are called *complementary* or *contradictory*. For instance, '$\sim$*guilty*' and '*guilty*' are two complementary literals. Atoms, while propositional in nature, will be represented in a manner similar to a first-order language. Also, variables will be considered only as place-holders for constants. Thus, an atom containing variables will be a schematic representation of all the possible atoms that can be obtained replacing the variables with the constants in the representation language (Lifschitz, 1996). Our notation will follow standard typographic conventions used in Logic Programming. To distinguish between atoms and variables, as usual, atoms will start with a lowercase letter and variables with an uppercase letter. For instance, '$\sim dangerous(X)$' and '$dangerous(X)$' are two schematic literals.

A *defeasible logic program*, abbreviated *d.l.p.*, is a set of facts, strict rules, and defeasible rules defined as follows:

- *Facts* are ground (objective) literals, e.g. *guilty*, $price(100)$, $\sim alive$. In DeLP facts are used for representing information that is considered to hold in the application domain. Hence, as it will be explained below, a 'valid' *d.l.p.* cannot contain two complementary facts.
- *Strict Rules* represent a relation between a ground literal L_0, or *head* of the rule, and a set of ground literals $\{L_i\}_{i>0}$, or *body* of the rule, and are denoted by $L_0 \leftarrow L_1, \ldots, L_n$; strict rules correspond syntactically to *basic rules* in Logic Programming (Lifschitz, 1996). The use of the adjective 'strict' emphasises that the relation between the head and the body of the rule is such that if the body is accepted then the head must also be accepted. The examples of strict rules shown below can be understood as expressing that: *if somebody is guilty then it is not innocent*, *cats are mammals*, and *if there are not many surfers then there are few surfers*:

$$\sim innocent \leftarrow guilty$$

$$mammal \leftarrow cat$$

$$few\_surfers \leftarrow \sim many\_surfers$$

- *Defeasible Rules* are used to represent a weaker connection between pieces of information, they are denoted by $L_0 \mathrel{-\!\!\prec} L_1, \ldots, L_n$, and like strict rules, the head of the rule L_0 is a ground literal and its body $\{L_i\}_{i>0}$ is a set of ground literals. Unlike strict rules, acceptance of the body of a defeasible rule does not always lead to the acceptance of the head. In defeasible rules, literals on the body can be preceded with the default negation symbol '*not*' (see discussion below). Examples of defeasible rules follows, the first one represents that *usually, mosquitoes are not dangerous*, and the second says that *reasons to believe mosquitoes are carrying dengue, justify the belief they are dangerous*:

$$\sim dangerous \mathrel{-\!\!\prec} mosquito$$

$$dangerous \mathrel{-\!\!\prec} mosquito, dengue$$

Note that, from a syntactic point of view, strict and defeasible rules differ only in the symbol between the head and the body of the rule. It is interesting to remark here that the representational choice between these two forms of relating the head and the body of a rule is ultimately a matter of context, sometimes a rule could change according to the environment in which it is used; for instance, a rule that locally can be considered strict could become defeasible in a larger environment.

Defeasible rules allow to represent a weak connection between the body (antecedent) and the head (consequence) of the rule. A defeasible rule $H \mathrel{-\!\!\prec} B$ expresses that reasons to believe in B provide a (defeasible) reason to believe in H. As an example, consider a scenario where an agent has to decide how to spend the day. Then, the defeasible rule '$nice \mathrel{-\!\!\prec} waves$' can represent that '*reasons to believe that there are big waves at the beach, is a reason to believe that it should be a nice day for surfing*'. The connection between '*waves*' and '*nice*' is weak in the sense that there might be other reasons such as '*normally, if it is raining it is not nice for surfing*', represented as '$\sim nice \mathrel{-\!\!\prec} rain$', that will lead to the contrary conclusion. Suppose that today there are big waves and it is raining, then the acceptance of the body of the rule '$nice \mathrel{-\!\!\prec} waves$' does not lead directly to the acceptance of the head.

Nevertheless, strict rules establish a strict connection between body and head; thus, the rule '$\sim working \leftarrow vacation$' represents the fact that in vacation an agent is not working. Then, as we will show below, due to this strict connection in DeLP if '*vacation*' is accepted then '$\sim working$' is also accepted.

As discussed in Alferes, Pereira, and Przymusinski (1996), logic programs, deductive databases, and more generally non-monotonic theories, use various forms of *default negation*, usually denoting '*not F*' as the default negation of '*F*', whose major distinctive feature is that *not* '*F*' is assumed by default, i.e. it is assumed in the absence of sufficient evidence to the contrary '*F*'. In DeLP default negation can be used in the body of defeasible rules; for instance, the rule '$alive \mathrel{-\!\!\prec} not\ dead$' represents that *if we cannot be sure that something is dead, then there are reasons to believe that it is alive*. Thus, DeLP allows for the use of two different forms of negation. For instance, the rule '$\sim guilty \mathrel{-\!\!\prec} not\ guilty$' represents that *if we cannot be sure that somebody is guilty, then there are reasons to believe that he is not guilty*; and the defeasible rule

$$\sim cross\_railway\_tracks \mathrel{-\!\!\prec} not \sim train\_is\_coming$$

can be understood as *if we cannot be sure the train is not coming then there are (defeasible) reasons to believe we should not be crossing the railway tracks*.

Methodologically, it is suggested that the default negation '*not*' be used only on literals appearing in the body of defeasible rules; these literals, that are known as *extended* literals, provide another form of defeasibility. The motivation for this recommendation is that strict rules with extended literals in their body would become defeasible (see García & Simari, 2004 for more details).

A defeasible rule with an empty body represents an interesting representational device that can be used as information that could be used when no contrary information exists; this type of defeasible rule was introduced in several approaches to defeasible reasoning and is called a *presumption* (García & Simari, 2004, Martínez et al., 2012, Nute, 1988).

Note that the symbols '$\prec\!\!\!-$' and '$\leftarrow$' denote meta-relations between a literal and a set of literals, and have no interaction with language symbols. As in Logic Programming, strict and defeasible rules are not conditionals nor implications, they are inference rules; consequently, it is not possible to apply contraposition to program rules, strict or defeasible. All defeasible logic programs are ground; nevertheless, following the usual convention in Logic Programming, some examples use 'schematic rules' with variables for notational convenience. These schematic rules represent all possible instantiations as ground rules using the constants appearing in the signature of the language of the program (Lifschitz, 1996).

A defeasible logic program is set of facts and rules, however, when required, a *d.l.p.* is denoted by (Π, Δ), to distinguish the subset Π of facts and strict rules and the subset Δ of defeasible rules. This denotational distinction will be useful for introducing central concepts below. Consider for instance the program $(\Pi_{2.1}, \Delta_{2.1})$ in the following example.

Example 2.1 DeLP-program $(\Pi_{2.1}, \Delta_{2.1})$

$$\Pi_{2.1} = \left\{ \begin{array}{l} monday \\ cloudy \\ dry\_season \\ waves \\ grass\_grown \\ hire\_gardener \\ vacation \\ \sim working \leftarrow vacation \\ few\_surfers \leftarrow \sim many\_surfers \\ \sim surf \leftarrow ill \end{array} \right\} \qquad \Delta_{2.1} = \left\{ \begin{array}{l} surf \prec\!\!\!- nice, spare\_time \\ nice \prec\!\!\!- waves \\ \sim nice \prec\!\!\!- rain \\ rain \prec\!\!\!- cloudy \\ \sim rain \prec\!\!\!- dry\_season \\ spare\_time \prec\!\!\!- \sim busy \\ \sim busy \prec\!\!\!- \sim working \\ cold \prec\!\!\!- winter \\ working \prec\!\!\!- monday \\ busy \prec\!\!\!- yard\_work \\ yard\_work \prec\!\!\!- grass\_grown \\ \sim yard\_work \prec\!\!\!- hire\_gardener \\ many\_surfers \prec\!\!\!- waves \\ \sim many\_surfers \prec\!\!\!- monday \end{array} \right\}$$

The program above shows an application example with several facts and rules. The set $\Pi_{2.1}$ has some facts representing that: *it is Monday*, *it is cloudy*, *it is the dry season*, *there are big waves at the beach*, *the grass of our yard is grown*, *we can hire a gardener*, *we are on vacation*, and three strict rules that represent: *on vacation we do not work*, *if there are not many surfers then there are few surfers* and *if you are ill then you should not go surfing*.

The set $\Delta_{2.1}$ contains defeasible rules that can be used in the process of building arguments for supporting or attacking different claims. For instance, the first rule in $\Delta_{2.1}$ represents that *having spare time on a nice day, is a good reason to go surfing*, and can be used in support of the claim *go surfing* (*surf*). Note that there is also a rule that can be used for concluding that *today is not a nice day* ($\sim$ *nice*) and another rule that provides reasons for concluding that *today I am busy* (*busy*). In the next section, we will show how rules and facts are used in defeasible derivations and then we will show what constitutes an argument that supports a conclusion.

3. Defeasible derivations and arguments

In DeLP a ground literal L will have a defeasible derivation from a program (Π, Δ), if there is a finite sequence of ground literals $L_1, L_2, \ldots, L_n = L$, where L_i is a fact in Π, or there exists a rule R_i in (Π, Δ) (strict or defeasible) with head L_i and body $B_1, B_2, \ldots, B_m$ such that every literal B_j, $1 \leq j \leq m$, of the body is either an element L_k already appearing in the sequence preceding L_i $(k < i)$, or an extended literal, i.e. a literal affected by default negation.

In the program $(\Pi_{2.1}, \Delta_{2.1})$ shown in Example 2.1, the literal *surf* has a defeasible derivation: *vacation*, $\sim$ *working*, $\sim$ *busy*, *spare_time*, *waves*, *nice*, *surf*, which contains two facts (*vacation* and *waves*), and the use of a strict rule ($\sim$*working* $\leftarrow$ *vacation*) and four defeasible rules. Note that every fact of a DeLP has a defeasible derivation; however, not every head of a rule has a derivation, for instance, neither *cold* nor $\sim$ *surf* have a defeasible derivation. If a literal L has a derivation that uses only uses facts and strict rules from Π and no defeasible rules, in this case we say that L has a *strict derivation*; thus, $\sim$ *working* has a strict derivation from $\Pi_{2.1}$. Note that literals that have a strict derivation must be facts or the head of a strict rules; however, a literal can be the head of a strict rule and might have a defeasible derivation, but not a strict derivation. For instance, *few_surfers* has no strict derivation from $\Pi_{2.1}$, although it has a defeasible derivation from $(\Pi_{2.1}, \Delta_{2.1})$ that uses a defeasible rule for the derivation of $\sim$*many_surfers*.

Since strong negation is allowed in the head of rules, it is possible to obtain defeasible derivations of contradictory literals from a program, e.g. both literals *rain* and $\sim$ *rain* have a defeasible derivation from $(\Pi_{2.1}, \Delta_{2.1})$. Observe that the defeasible derivation for *rain* depends on the use of the defeasible rule *rain* $\prec$ *cloudy* whereas the defeasible derivation of $\sim$ *rain* uses the defeasible rule $\sim$*rain* $\prec$ *dry_season*. It is important to note that in DeLP the set Π is used to represent non-defeasible information, consequently it is required that the set be representationally coherent. Therefore, for any program we assume that no pair of contradictory literals can be derived from Π, i.e. no strict derivation for contradictory literals can be obtained from a DeLP-program.

In the presence of defeasible derived contradictory literals, DeLP uses a dialectical process to decide which information prevails, i.e. information such that no acceptable reason can be put forward against it. Reasons will be supported by arguments which are structures based in defeasible derivations. The notion of argument is introduced by the following definition.

DEFINITION 3.1 Let (Π, Δ) be a defeasible logic program and L a ground literal. We say that $\mathcal{A}$ is an argument for the conclusion L from (Π, Δ), denoted by $\langle \mathcal{A}, L\rangle$, if $\mathcal{A}$ is a minimal set of defeasible rules $(\mathcal{A} \subseteq \Delta)$, such that: (1) there exists a defeasible derivation for L from $\Pi \cup \mathcal{A}$, (2) no pair of contradictory literals can be defeasibly derived from $\Pi \cup \mathcal{A}$, and (3). If $\mathcal{A}$ contains some rule with an extended literal '*not F*', then the literal F can not be in the defeasible derivation of L from $\Pi \cup \mathcal{A}$.

Observe that although facts and strict rules are used in the defeasible derivation, the argument structure only mentions the defeasible rules. If there exists an argument $\mathcal{A}$ for a literal L then we will also say that $\mathcal{A}$ supports L.

Example 3.2 From program $(\Pi_{2.1}, \Delta_{2.1})$ introduced in Example 2.1 we can obtain several arguments:

$$\langle \mathcal{A}_0, \mathit{surf}\rangle = \left\langle \left\{ \begin{array}{l} \mathit{surf} \prec \mathit{nice}, \mathit{spare\_time} \\ \mathit{nice} \prec \mathit{waves} \\ \mathit{spare\_time} \prec {\sim}\mathit{busy} \\ {\sim}\mathit{busy} \prec {\sim}\mathit{working} \end{array} \right\}, \mathit{surf} \right\rangle$$
$$\langle \mathcal{A}_1, {\sim}\mathit{nice}\rangle = \langle \{({\sim}\mathit{nice} \prec \mathit{rain}); (\mathit{rain} \prec \mathit{cloudy})\}, {\sim}\mathit{nice}\rangle$$

$\langle \mathcal{A}_2, nice\rangle = \langle\{nice \prec waves\}, nice\rangle$
$\langle \mathcal{A}_3, rain\rangle = \langle\{rain \prec cloudy\}, rain\rangle$
$\langle \mathcal{A}_4, {\sim}rain\rangle = \langle\{{\sim}rain \prec dry\_season\}, {\sim}rain\rangle$
$\langle \mathcal{A}_5, busy\rangle = \langle\{(busy \prec yard\_work); (yard\_work \prec grass\_grown)\}, busy\rangle$
$\langle \mathcal{A}_6, {\sim}yard\_work\rangle = \langle\{{\sim}yard\_work \prec hire\_gardener\}, {\sim}yard\_work\rangle$
$\langle \mathcal{A}_7, yard\_work\rangle = \langle\{yard\_work \prec grass\_grown\}, yard\_work\rangle$
$\langle \mathcal{A}_8, spare\_time\rangle = \langle\{(spare\_time \prec {\sim}busy); ({\sim}busy \prec {\sim}working)\}, spare\_time\rangle$
$\langle \mathcal{A}_9, {\sim}busy\rangle = \langle\{{\sim}busy \prec {\sim}working\}, {\sim}busy\rangle$
$\langle \mathcal{A}_{10}, many\_surfers\rangle = \langle\{many\_surfers \prec waves\}, many\_surfers\rangle$
$\langle \mathcal{A}_{11}, {\sim}many\_surfers\rangle = \langle\{{\sim}many\_surfers \prec monday\}, {\sim}many\_surfers\rangle$

We can see that from the program $(\Pi_{2.1}, \Delta_{2.1})$ there is argument $(\mathcal{A}_0)$ for the conclusion *surf*; however, observe that there are arguments for *nice* and $\sim$ *nice*, and arguments for *busy* and $\sim$ *busy*. All the arguments rely on the same facts.

Arguments are related in different forms, and two interesting relations are the subargument relation and counterargument relation. But first, we will make some observations regarding the definition of argument.

As stated in Definition 3.1, a literal L has a supporting argument $\mathcal{A}$ from a program (Π, Δ) if $\mathcal{A}$ is a minimal set of defeasible rules such that there exist a defeasible derivation for L from $\Pi \cup \mathcal{A}$, and $\Pi \cup \mathcal{A}$ is not contradictory. Therefore, it could happen that a literal L has a defeasible derivation from a program $\mathcal{P}$ but there is no argument for L from $\mathcal{P}$, as in the following program:

$$\Pi_{3.2} = \left\{\begin{array}{l} night \\ at\_market \\ {\sim}at\_home \leftarrow at\_market \end{array}\right\} \qquad \Delta_{3.2} = \{at\_home \prec night\}$$

Here, there exists a defeasible derivation for *at_home* from $(\Pi_{3.2}, \Delta_{3.2})$, but the set $\Pi_{3.2} \cup \{at\_home \prec night\}$ is contradictory, and for that reason there is no argument for *at_home*.

Let us consider again the program $(\Pi_{2.1}, \Delta_{2.1})$, and the following subset of defeasible rules $S = \{nice \prec waves, {\sim}busy \prec {\sim}working\}$ $(S \subseteq \Delta_{2.1})$. Observe that $S \cup \Pi_{2.1}$ is not contradictory and allows for the defeasible derivation of *nice*; nevertheless S is not an argument for *nice* because it is not minimal. In Example 3.2 we have shown that $\mathcal{A}_2 \subset S$ is an argument for *nice*.

Next, we will describe the subargument and counterargument relations. In the following section these relations will be used in the introduction of an inference mechanism.

Given a *d.l.p.* (Π, Δ) and two arguments $\langle \mathcal{A}, L\rangle$ and $\langle \mathcal{B}, Q\rangle$ obtained from it, if $\mathcal{B} \subseteq \mathcal{A}$ then we say that $\langle \mathcal{B}, Q\rangle$ is a *subargument* of $\langle \mathcal{A}, L\rangle$ and that $\langle \mathcal{A}, L\rangle$ is the *superargument* of $\langle \mathcal{B}, Q\rangle$ (note that trivially every argument is a subargument/superargument of itself). For instance, consider the Example above, $\langle \mathcal{A}_3, rain\rangle$ is a subargument of $\langle \mathcal{A}_1, {\sim}nice\rangle$, and $\langle \mathcal{A}_2, nice\rangle$ is a subargument of $\langle \mathcal{A}_0, surf\rangle$.

Two literals L and Q are said to disagree in the context of the program (Π, Δ) if the set $\Pi \cup \{L, Q\}$ is contradictory, i.e. it is possible to strictly derive a literal and its complementary. For instance, in the program $(\Pi_{2.1}, \Delta_{2.1})$ the literals *working* and *vacation* disagree because from $\Pi_{2.1} \cup \{working, vacation\}$ it is possible to strictly derive ${\sim}working$. As a particular case, two literals disagree if they are contradictory (e.g. *nice* and ${\sim}nice$).

In DeLP, an argument $\langle \mathcal{B}, Q\rangle$ is a *counterargument* for $\langle \mathcal{A}, L\rangle$at literal P, if there exists a subargument $\langle \mathcal{C}, P\rangle$ of $\langle \mathcal{A}, L\rangle$ such that P and Q *disagree*. The literal P is referred to as the *counterargument point* and $\langle \mathcal{C}, P\rangle$ as the *disagreement subargument*. If $\langle \mathcal{B}, Q\rangle$ is a counterargument for $\langle \mathcal{A}, L\rangle$, then we also say that $\langle \mathcal{B}, Q\rangle$ *attacks* $\langle \mathcal{A}, L\rangle$, and that $\langle \mathcal{B}, Q\rangle$ and $\langle \mathcal{A}, L\rangle$ are in *conflict*.

Consider the program $(\Pi_{2.1}, \Delta_{2.1})$. $\langle \mathcal{A}_1, {\sim}nice\rangle$ is a counterargument for $\langle \mathcal{A}_2, nice\rangle$ and viceversa because in this particular case the conclusion of both arguments disagree. As another example,

$\langle \mathcal{A}_4, \sim rain \rangle$ is a counterargument for $\langle \mathcal{A}_1, \sim nice \rangle$ at the counterargument point *rain* and $\langle \mathcal{A}_3, rain \rangle$ is the disagreement subargument. Note that in DeLP a counter-argument for an argument $\mathcal{A}$ is also a counter-argument for any super-argument of $\mathcal{A}$. For instance, from $(\Pi_{2.1}, \Delta_{2.1})$ since $\langle \mathcal{A}_1, \sim nice \rangle$ is a counter-argument for $\langle \mathcal{A}_2, nice \rangle$ and $\langle \mathcal{A}_0, surf \rangle$ is a super-argument of $\langle \mathcal{A}_2, nice \rangle$, then $\langle \mathcal{A}_1, \sim nice \rangle$ is a counter-argument for $\langle \mathcal{A}_0, surf \rangle$.

Given a program $\mathcal{P}$, every literal L that has a strict derivation from $\mathcal{P}$, has a supporting argument with no defeasible rules (i.e. $\mathcal{A} = \emptyset$) that supports L. Also note that in DeLP there is no possible counterargument for a claim having a strict derivation, see García and Simari (2004) for the proof. It was shown in an example above, involving the program $(\Pi_{3.2}, \Delta_{3.2})$, that there is a strict derivation for $\sim$*at_home*; hence, for that reason there is no argument for *at_home* and therefore no counter-argument can exist. Observe also that from $(\Pi_{2.1}, \Delta_{2.1})$ there is a strict derivation for $\sim$*working*, however, although there is a derivation for *working*, no argument for *working* can exist, and hence, no counter-argument for $\sim$*working*.

4. Warrants and answers for queries

A query is issued to a defeasible logic program (Π, Δ) in the form of a ground literal Q, and this query will succeed when Q can be *warranted* from (Π, Δ). The intuitive meaning of a literal being warranted is that it is supported by an argument that has no counterargument still standing against it. As it is discussed below, there are four possible answers for a query Q posed to a program $\mathcal{P}$: YES, if Q is warranted from $\mathcal{P}$; NO, if the complement of Q is warranted from $\mathcal{P}$; UNDECIDED, if neither Q nor its complement is warranted from $\mathcal{P}$; and UNKNOWN, if Q is not in the signature of the program $\mathcal{P}$. For instance, as we will show later in this section in detail, from the program $(\Pi_{2.1}, \Delta_{2.1})$ the answer for '*surf*' is YES, the answer for '*busy*' is NO, the answer for '*many_surfers*' is UNDECIDED, and the answer for '*beach_closed*' is UNKNOWN.

Informally, a literal Q will be warranted if there exists at least an argument $\mathcal{A}$ supporting Q that prevails after going through a dialectical process considering all its counterarguments. To asses this, the counterarguments must be compared to the supporting argument, and each counterargument that is in some predetermined sense found better, becomes a potential *defeater* which in turn is analysed in a similar way. In the rest of this section we will be describing all the elements involved in the warranting process.

Let us consider again the program $(\Pi_{2.1}, \Delta_{2.1})$. The argument $\langle \mathcal{A}_1, \sim nice \rangle$ is a counterargument for $\langle \mathcal{A}_2, nice \rangle$ and viceversa. As we will show, if a preference between $\mathcal{A}_1$ and $\mathcal{A}_2$ can be established, then one of these arguments will *defeat* the other and its conclusion will prevail. In DeLP a *defeater relation* is defined in terms of counter-argument and an argument comparison criterion as follows. Recall that if an argument $\mathcal{C}$ counter-arguments $\mathcal{A}$, then there exists a disagreement sub-argument $\mathcal{B}$ of $\mathcal{A}$.

Given a preference criterion, an argument $\mathcal{D}$ is a *defeater* for $\mathcal{A}$, if one of the following conditions holds:

(a) $\mathcal{D}$ is a counter-argument for $\mathcal{A}$ with disagreement sub-argument $\mathcal{B}$, and $\mathcal{D}$ is preferred to $\mathcal{B}$, in this case $\mathcal{D}$ is a called a *proper defeater* for $\mathcal{A}$, or
(b) $\mathcal{D}$ is a counter-argument for $\mathcal{A}$ with disagreement sub-argument $\mathcal{B}$, and $\mathcal{D}$ is equally preferred to, or $\mathcal{D}$ is incomparable with $\mathcal{B}$, in this case $\mathcal{D}$ is called a *blocking defeater* for $\mathcal{A}$, or
(c) $\mathcal{D}$ is argument for the conclusion F and '*not F*' is in some rule of $\mathcal{A}$, in this case $\mathcal{D}$ is called an attack to an assumption of $\mathcal{A}$.

Observe that in the case that $\mathcal{B}$ is preferred to $\mathcal{D}$, although $\mathcal{D}$ remains a counterargument for $\mathcal{A}$, $\mathcal{D}$ is not a defeater for $\mathcal{A}$. As an argument trivially is a subargument of itself, it is possible that the disagreement subargument be $\mathcal{A}$ itself, thus, in this case $\mathcal{D}$ will be compared directly with $\mathcal{A}$.

In the context of program $(\Pi_{2.1}, \Delta_{2.1})$, $\langle \mathcal{A}_4, {\sim}rain \rangle$ is a counterargument for $\langle \mathcal{A}_1, {\sim}nice \rangle$ at the counterargument point *rain* and $\langle \mathcal{A}_3, rain \rangle$ is the disagreement subargument; therefore, $\mathcal{A}_4$ is compared with $\mathcal{A}_3$ to determine if it is a defeater.

The argument comparison criterion is modular in DeLP; hence, it is possible to use any preference criterion established over the set of arguments. This allows the user to select the most appropriate criterion for the application domain that is being represented. In García (2000), García and Simari (2004), and Ferretti, Errecalde, García, and Simari (2008), different argument comparison criteria were defined. In the examples of this paper we will use a comparison criterion that was introduced in García and Simari (2004). This argument preference relation uses a partial order defined over defeasible rules that is denoted by '$<_R$'. The intuitive meaning of '$R_1 <_R R_2$' is that: *the rule R_2 is preferred over R_1 in the application domain described by the program.*

In this argument comparison criterion, referred to as *rule's priorities*, an argument $\mathcal{A}_1$ is preferred to $\mathcal{A}_2$ if the following two conditions hold:

(1) there exists at least a rule $r_a \in \mathcal{A}_1$ and a rule $r_b \in \mathcal{A}_2$ such that $r_b <_R r_a$,
(2) and there is no rule $r'_b \in \mathcal{A}_2$, and $r'_a \in \mathcal{A}_1$ such that $r'_a <_R r'_b$.

Example 4.1 Consider the program $(\Pi_{2.1}, \Delta_{2.1})$ introducing the following priorities over its rules:

$$(rain \prec cloudy) <_R ({\sim}rain \prec dry\_season)$$

$$(yard\_work \prec grass\_grown) <_R ({\sim}yard\_work \prec hire\_gardener)$$

Lets discuss the relations between some of the arguments built from that program:

$$\langle \mathcal{A}_0, surf \rangle = \left\langle \left\{ \begin{array}{l} surf \prec nice, spare\_time \\ nice \prec waves \\ spare\_time \prec {\sim}busy \\ {\sim}busy \prec {\sim}working \end{array} \right\}, surf \right\rangle$$

$$\begin{aligned} \langle \mathcal{A}_1, {\sim}nice \rangle &= \langle \{({\sim}nice \prec rain); (rain \prec cloudy)\}, {\sim}nice \rangle \\ \langle \mathcal{A}_3, rain \rangle &= \langle \{rain \prec cloudy\}, rain \rangle, \text{and} \\ \langle \mathcal{A}_4, {\sim}rain \rangle &= \langle \{{\sim}rain \prec dry\_season\}, {\sim}rain \rangle, \\ \langle \mathcal{A}_5, busy \rangle &= \langle \{(busy \prec yard\_work); (yard\_work \prec grass\_grown)\}, busy \rangle \end{aligned}$$

Then, $\langle \mathcal{A}_4, {\sim}rain \rangle$ is preferred over $\langle \mathcal{A}_3, rain \rangle$. Since $\langle \mathcal{A}_4, {\sim}rain \rangle$ is a counterargument for $\langle \mathcal{A}_1, {\sim}nice \rangle$ with $\langle \mathcal{A}_3, rain \rangle$ as the disagreement subargument, then $\langle \mathcal{A}_4, {\sim}rain \rangle$ is a proper defeater for $\langle \mathcal{A}_1, {\sim}nice \rangle$. Note that $\langle \mathcal{A}_4, {\sim}rain \rangle$ is a also a proper defeater for $\langle \mathcal{A}_3, rain \rangle$. Taking into account that $\langle \mathcal{A}_1, {\sim}nice \rangle$ is a counterargument for $\langle \mathcal{A}_2, nice \rangle$ and viceversa, and $\langle \mathcal{A}_1, {\sim}nice \rangle$ and $\langle \mathcal{A}_2, nice \rangle$ are not comparable in this criterion, it results that $\langle \mathcal{A}_1, {\sim}nice \rangle$ is a blocking defeater for $\langle \mathcal{A}_2, nice \rangle$ and viceversa.

Observe that the argument $\langle \mathcal{A}_0, surf \rangle$ has two blocking defeaters: $\langle \mathcal{A}_1, {\sim}nice \rangle$ at the counterargument point *nice* and $\langle \mathcal{A}_5, busy \rangle$ at the counterargument point ${\sim}busy$. Since $\langle \mathcal{A}_6, {\sim}yard\_work \rangle$ is preferred to $\langle \mathcal{A}_7, yard\_work \rangle$ then $\langle \mathcal{A}_6, {\sim}yard\_work \rangle$ is a proper defeater for both $\langle \mathcal{A}_7, yard\_work \rangle$ and $\langle \mathcal{A}_5, busy \rangle$. Finally, $\langle \mathcal{A}_{10}, many\_surfers \rangle$ is a blocking defeater for $\langle \mathcal{A}_{11}, {\sim}many\_surfers \rangle$ and viceversa.

If an argument $\langle \mathcal{A}_1, L_1 \rangle$ is defeated by $\langle \mathcal{A}_2, L_2 \rangle$, then $\langle \mathcal{A}_2, L_2 \rangle$ represents a reason for rejecting $\langle \mathcal{A}_1, L_1 \rangle$; nevertheless, as $\langle \mathcal{A}_2, L_2 \rangle$ also is an argument, a defeater $\langle \mathcal{A}_3, L_3 \rangle$ for it might also exist, defeating $\langle \mathcal{A}_2, L_2 \rangle$ and reinstating $\langle \mathcal{A}_1, L_1 \rangle$. This analysis is repeated and the argument $\langle \mathcal{A}_3, L_3 \rangle$ could in turn become defeated, reinstating $\langle \mathcal{A}_2, L_2 \rangle$, and so on. In this manner, a sequence of arguments (called argumentation line) is built; in it, each argument is a defeater of its predecessor in the sequence.

More formally, given $\langle \mathcal{A}_1, L_1 \rangle$, an *argumentation line* for $\langle \mathcal{A}_1, L_1 \rangle$ is a sequence of arguments, denoted by $\Lambda = [\langle \mathcal{A}_1, L_1 \rangle, \langle \mathcal{A}_2, L_2 \rangle, \langle \mathcal{A}_3, L_3 \rangle, \ldots]$, where each element of the sequence $\langle \mathcal{A}_i, L_i \rangle$, $i > 1$, is a defeater of its predecessor $\langle \mathcal{A}_{i-1}, L_{i-1} \rangle$. The first element, $\langle \mathcal{A}_1, L_1 \rangle$, becomes a *supporting* argument for L_1, $\langle \mathcal{A}_2, L_2 \rangle$ an *interfering* argument, $\langle \mathcal{A}_3, L_3 \rangle$ a supporting argument, $\langle \mathcal{A}_4, L_4 \rangle$ an interfering one, continuing in that manner. Thus, an argumentation line can be split into two disjoint sets: $\Lambda_S = \{\langle \mathcal{A}_1, L_1 \rangle, \langle \mathcal{A}_3, L_3 \rangle, \langle \mathcal{A}_5, L_5 \rangle, \ldots\}$ of supporting arguments, and $\Lambda_I = \{\langle \mathcal{A}_2, L_2 \rangle, \langle \mathcal{A}_4, L_4 \rangle, \ldots\}$ of interfering arguments. For instance, from the program $(\Pi_{2.1}, \Delta_{2.1})$ the following argumentation line can be obtained $[\langle \mathcal{A}_0, \mathit{surf} \rangle, \langle \mathcal{A}_1, {\sim}\mathit{nice} \rangle, \langle \mathcal{A}_4, {\sim}\mathit{rain} \rangle]$, where $\mathcal{A}_1$ is an interfering argument for *surf*, and $\mathcal{A}_4$ is a supporting argument for *surf*.

There are undesirable situations, analysed in the literature, that can occur leading to an infinite argumentation line; for instance, if an argument is reintroduced in the sequence, the process could repeat itself indefinitely. Circular argumentation and other forms of *fallacious argumentation* have been studied in detail (see García & Simari, 2004, Simari et al., 1994b).

In DeLP, argumentation lines are required to be *acceptable*, that is, they have to be finite, an argument cannot appear twice in it, and the set of supporting (resp. interfering) arguments in the line have to be *concordant*. Given a program (Π, Δ), a set of arguments $\{\langle \mathcal{A}_i, L_i \rangle\}_{i=1}^{n}$ is concordant if it is not possible to have a defeasible derivation for a pair of contradictory literals from the set $\Pi \cup \bigcup_{i=1}^{n} \mathcal{A}_i$. For instance, the set $\{\langle \mathcal{A}_0, \mathit{surf} \rangle, \langle \mathcal{A}_4, {\sim}\mathit{rain} \rangle\}$ is concordant, whereas the set $\{\langle \mathcal{A}_0, \mathit{surf} \rangle, \langle \mathcal{A}_1, {\sim}\mathit{nice} \rangle\}$ is not because from $\Pi \cup \mathcal{A}_0 \cup \mathcal{A}_1$ there are defeasible derivations for *nice* and ~*nice*.

The existence of blocking defeaters introduces another situation that is addressed by the last requirement; that is, after introducing a blocking defeater in a line, the next defeater is required to be a proper one to reinstate the blocked argument. Formally, an argumentation line $\Lambda = [\langle \mathcal{A}_1, L_1 \rangle, \ldots \langle \mathcal{A}_n, L_n \rangle]$ is *acceptable* if and only if:

(1) Λ is a finite sequence.
(2) The set Λ_S of supporting arguments (resp. Λ_I) is concordant.
(3) No argument $\langle \mathcal{A}_k, L_k \rangle$ in Λ is a disagreement subargument of an argument $\langle \mathcal{A}_i, L_i \rangle$ appearing earlier in Λ, $i < k$.
(4) For all i, such that $\langle \mathcal{A}_i, L_i \rangle$ is a blocking defeater for $\langle \mathcal{A}_{i-1}, L_{i-1} \rangle$, if $\langle \mathcal{A}_{i+1}, L_{i+1} \rangle$ exists, then $\langle \mathcal{A}_{i+1}, L_{i+1} \rangle$ is a proper defeater for $\langle \mathcal{A}_i, L_i \rangle$.

For instance, from the program $(\Pi_{2.1}, \Delta_{2.1})$ two acceptable argumentation lines for $\langle \mathcal{A}_0, \mathit{surf} \rangle$ can be obtained:

$$\Lambda_1 == [\langle \mathcal{A}_0, \mathit{surf} \rangle, \langle \mathcal{A}_1, {\sim}\mathit{nice} \rangle, \langle \mathcal{A}_4, {\sim}\mathit{rain} \rangle],$$

$$\Lambda_2 = [\langle \mathcal{A}_0, \mathit{surf} \rangle, \langle \mathcal{A}_5, \mathit{busy} \rangle, \langle \mathcal{A}_6, {\sim}\mathit{yard\_work} \rangle].$$

This example shows how, given a program, there can be more than one argumentation line starting with the same argument $\langle \mathcal{A}, L \rangle$. Therefore, analysing a single acceptable argumentation line for $\langle \mathcal{A}, L \rangle$ will not be enough to determine whether $\langle \mathcal{A}, L \rangle$ is an undefeated argument. In the general situation, there might be several defeaters $\langle \mathcal{B}_1, Q_1 \rangle, \langle \mathcal{B}_2, Q_2 \rangle, \ldots, \langle \mathcal{B}_k, Q_k \rangle$ for $\langle \mathcal{A}_1, L_1 \rangle$, and for each defeater $\langle \mathcal{B}_i, Q_i \rangle$ there could be in turn several defeaters; thus, a tree structure is defined which is called a dialectical tree. In this tree, the root is labelled with $\langle \mathcal{A}, L \rangle$ and every node (except the root) represents a defeater (proper or blocking) of its parent. Each branch in the tree, i.e. each path from a leaf to the root, corresponds to a different acceptable argumentation line.

Let $\langle \mathcal{A}_1, L_1 \rangle$ be an argument obtained from a DeLP-program $\mathcal{P}$, a *dialectical tree* for $\langle \mathcal{A}_1, L_1 \rangle$ from $\mathcal{P}$ is denoted by $\mathcal{T}_{\langle \mathcal{A}_1, L_1 \rangle}$ and is constructed as follows:

(1) The root of the tree is labelled with $\langle \mathcal{A}_1, L_1 \rangle$.
(2) Let N be a node labelled $\langle \mathcal{A}_n, L_n \rangle$, and $[\langle \mathcal{A}_1, L_1 \rangle, \ldots, \langle \mathcal{A}_n, L_n \rangle]$ be the sequence of labels of the path from the root to N. Let $\{\langle \mathcal{B}_1, Q_1 \rangle, \langle \mathcal{B}_2, Q_2 \rangle, \ldots, \langle \mathcal{B}_k, Q_k \rangle\}$ be the set of all the defeaters for $\langle \mathcal{A}_n, L_n \rangle$ from $\mathcal{P}$. For each defeater $\langle \mathcal{B}_i, Q_i \rangle$ $(1 \le i \le k)$, such that the argumentation line $\Lambda' = [\langle \mathcal{A}_1, L_1 \rangle, \ldots, \langle \mathcal{A}_n, L_n \rangle, \langle \mathcal{B}_i, Q_i \rangle]$ is acceptable, the node N has a child N_i labelled $\langle \mathcal{B}_i, Q_i \rangle$. If there is no defeater for $\langle \mathcal{A}_n, L_n \rangle$ or there is no $\langle \mathcal{B}_i, Q_i \rangle$ such that Λ' is acceptable, then N is a leaf.

Dialectical trees will be depicted as a tree where labelled triangles represent arguments and edges denote the defeat relation. An argument $\langle \mathcal{A}, L \rangle$ will be depicted as a triangle, where its upper vertex is labelled with L, and $\mathcal{A}$ is showed inside the triangle. Figure 1 shows the dialectical tree for argument $\langle \mathcal{A}_0, \mathit{surf} \rangle$ obtained from program $(\Pi_{2.1}, \Delta_{2.1})$. A small triangle inside a bigger one represents a subargument, and usually is presented in that way to show a disagreement subargument with the counterargument point where an edge ends. We will use solid edge with the arrowhead pointing from the proper defeater to the defeated argument, and edges with double pointed arrows to represent blocking defeaters.

A dialectical tree provides a useful structure for considering all possible acceptable argumentation lines that can be generated for deciding whether an argument is defeated. We call this tree *dialectical* because it represents an exhaustive dialectical analysis for the argument in its root.

Given a literal L and an argument $\langle \mathcal{A}, L \rangle$, to decide whether the literal L is warranted, every node in the dialectical tree $\mathcal{T}_{\langle \mathcal{A}, L \rangle}$ is recursively marked as "*D*" (*defeated*) or "*U*" (*undefeated*), obtaining a marked dialectical tree $\mathcal{T}^*_{\langle \mathcal{A}, L \rangle}$. Nodes are marked by a bottom-up procedure that starts marking all leaves in $\mathcal{T}^*_{\langle \mathcal{A}, L \rangle}$ as "*U*"s. Then, for each inner node $\langle \mathcal{B}, Q \rangle$ of $\mathcal{T}^*_{\langle \mathcal{A}, L \rangle}$, either:

(a) $\langle \mathcal{B}, Q \rangle$ will be marked as "*U*" iff every child of $\langle \mathcal{B}, Q \rangle$ is marked as "*D*", or
(b) $\langle \mathcal{B}, Q \rangle$ will be marked as "*D*" iff it has at least a child marked as "*U*".

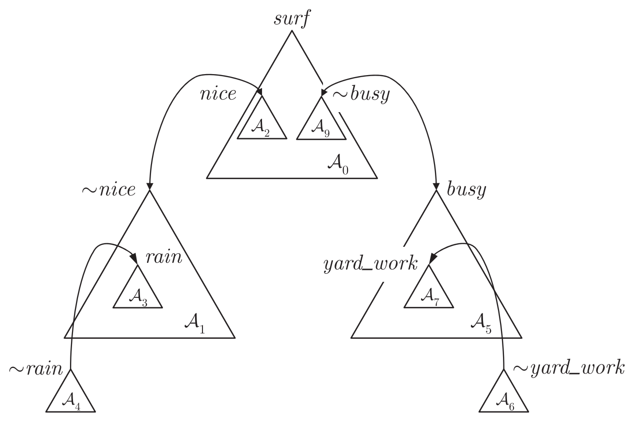

Figure 1. Dialectical tree $\mathcal{T}_{\langle \mathcal{A}_0, \mathit{surf} \rangle}$.

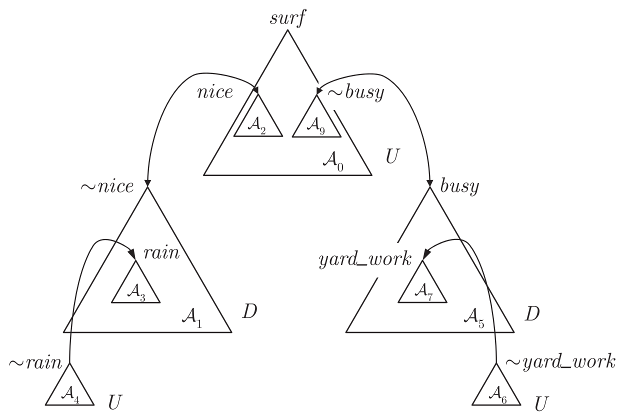

Figure 2. Marked dialectical tree $\mathcal{T}^*_{\langle\mathcal{A},L\rangle}$.

Marked dialectical trees will be depicted as a tree where the nodes are triangles decorated with the result of the marking procedure and the directed edges denote the defeat relation. Figure 2 shows the marked dialectical tree for argument $\langle\mathcal{A}_0, surf\rangle$ of program $(\Pi_{2.1}, \Delta_{2.1})$.

Given an argument $\langle\mathcal{A}, L\rangle$ obtained from a program $\mathcal{P}$, we will write $Mark(\mathcal{T}^*_{\langle\mathcal{A},L\rangle}) = U$ to denote that the root of $\mathcal{T}^*_{\langle\mathcal{A},L\rangle}$ is marked as "U"; otherwise we will write $Mark(\mathcal{T}^*_{\langle\mathcal{A},L\rangle}) = D$ (if the root of $\mathcal{T}^*_{\langle\mathcal{A},L\rangle}$ is marked as "D"). Thus, we can define warrant in terms of the marking procedure *Mark*:

Definition 4.2 Let $\mathcal{P}$ be a DeLP-program and L a DeLP-query. We say that $\mathcal{T}^*_{\langle\mathcal{A},L\rangle}$ warrants L and that L is warranted from $\mathcal{P}$ if $Mark(\mathcal{T}^*_{\langle\mathcal{A},L\rangle}) = U$.

A ground literal L is considered to be *warranted* if an argument $\mathcal{A}$ for L exists, and all the defeaters for $\langle\mathcal{A}, L\rangle$ are defeated. Therefore, this marking procedure provides an effective way of determining if a DeLP-query L is warranted. It is important to note that given a DeLP-query L, there can be several arguments that support L; therefore, L will be warranted if there exists at least one argument $\mathcal{A}$ for L such that the root of a dialectical tree for $\langle\mathcal{A}, L\rangle$ is marked as "U".

It has been observed in Chesñevar and Simari (2007) that the order in which the defeaters are considered is irrelevant to the final result of the marking process, only the argumentation lines are important for the outcome. The different tree configurations obtained from the set of argumentation lines (called bundle set) lead to the same marking of the root.

In García (2000) and García and Simari (2004) it was proved that in DeLP any claim C that has a strict derivation is warranted. This is so because any literal C that has a strict derivation from a program (Π, Δ) has no posible counterargument voiding the possibility that a defeater could exist. The reason for this is that no argument that disagrees with C could be build because the definition of argument requires it to be non-contradictory with Π.

The interpreter of DeLP takes a program $\mathcal{P}$, and a DeLP-query L as input. Then returns one of the following four possible answers: YES, if L is warranted from $\mathcal{P}$; NO, if the complement of L is warranted from $\mathcal{P}$; UNDECIDED, if neither L nor its complement are warranted from $\mathcal{P}$; or UNKNOWN, if L is not in the language of the program $\mathcal{P}$.

As an illustrating example, from the program $(\Pi_{2.1}, \Delta_{2.1})$ introduced in Example 2.1, the following answers can be obtained:

Query	Answer
$surf$	YES
$\sim surf$	NO
$many\_surfers$	UNDECIDED
$\sim many\_surfers$	UNDECIDED
$nice$	YES
$\sim nice$	NO
$beach\_closed$	UNKNOWN
$\sim working$	YES
$working$	NO
$few\_surfers$	UNDECIDED

Observe that in the table above the literal *beach_closed* has the answer UNKNOWN, this is so because the literal *beach_closed* is not present in the signature of $(\Pi_{2.1}, \Delta_{2.1})$. The answer for both ∼*many_surfers* and *many_surfers* is UNDECIDED, because as it was shown in Example 4.1 the argument $\langle \mathcal{A}_{10}, many\_surfers \rangle$ is a blocking defeater for $\langle \mathcal{A}_{11}, \sim many\_surfers \rangle$ and vice versa. Note that the literal ∼*working* is the head of a strict rule and also has a strict derivation, therefore, the answer is YES. Finally, observe that *few_surfers* is the head of a strict rule but has no strict derivation, because the body of that particular strict rule can not be strictly derived from $(\Pi_{2.1}, \Delta_{2.1})$. In this case, *few_surfers* has a defeasible derivation (defeasible rules are used in its derivation), and there is also an argument supporting it, but since there is a blocking defeater for that argument, no warrant for *few_surfers* is possible and the answer is UNDECIDED.

5. DeLP reasoning servers and contextual queries

In MASs, reasoning can be seen as a service that can be offered as part of a knowledge-based infrastructure providing a mechanism to exploit it. In this section we describe DeLP Servers, that are the support of argumentative reasoning services conceived for a MAS setting. In such an environment, DeLP-Servers, introduced in García et al. (2007), provide client agents that can be distributed in remote hosts with the ability to consult different reasoning services implemented as DeLP-Servers that can be distributed themselves. Figure 3 schematically depicts a situation with three servers and five client agents distributed in four hosts.

Common (or public) knowledge can be stored in a server and represented as a DeLP-program; then, client agents may send queries to a DeLP-server and receive the corresponding answer. To respond to a query, a DeLP-server uses the knowledge stored in it; but, the server offers the possibility of integrating the knowledge stored in it with 'private' knowledge the client making the query has the possibility to send as part of the query.

For example, consider a DeLP-server developed for a real-state application domain. A program can be stored in that server with some facts that represent the available properties for renting, and some defeasible rules that represent common knowledge modelling how to select a suited property. Then, client agents can consult for a property and submit, together with the query, some private information that represents its current situation or requirements (e.g. *has children, works at home, has no car, cannot afford a rent of* \$3000 *or more*). If that is the case, this private knowledge is added temporarily to the knowledge stored in the server giving a particular *context* for the query. This context is private knowledge that the server will use for answering the query and

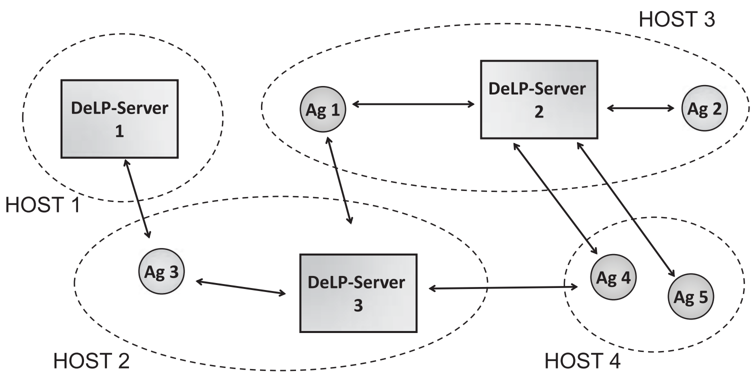

Figure 3. Agents can query different services with different knowledge bases.

will not affect other future queries. Thus, a client agent cannot make permanent changes to the public DeLP-program stored in a server. The temporal scope of the contextual information sent in a query is limited and it will disappear when the process performed by the DeLP-server to answer that query finishes. Continuing with our example, one client, *Client*$_1$, can consult if some property *p* is suited for her, providing the context that she is *single* and *has a car*. Other client, *Client*$_2$, can consult if *p* is suited for him, providing a different context: he *has two children* and *work downtown*. Since queries do not affect permanent changes in the program stored in a server, both contextual queries can be received and processed simultaneously, and one will not affect the answer of the other.

Below the concept of *contextual query* will be formalised. We will show that a contextual query provides to the server a particular DeLP-query, some contextual information in the form of DeLP-program and the information of how to integrate this contextual information with the public program stored in the server. To answer contextual queries, a DeLP-server uses both the common knowledge stored in it, and the context that clients can send attached to a query. The conceptual structure is described in Figure 4.

Several agents can consult the same DeLP-server, and the same agent can consult several DeLP-Servers, as shown in Figure 3. For instance, in a particular medical application, dedicated DeLP-Servers can maintain different programs, supporting particular shared knowledge of specific specialties of the medical profession. Hence, an agent can pose the same contextual query to different servers, and obtain a different answer from each of the servers. Our approach does not

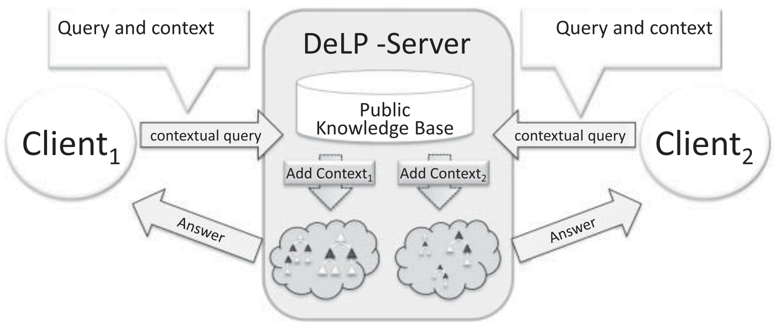

Figure 4. Agents can query a DeLP-server creating a context.

impose any restriction over the type, architecture, or implementation language of the client agents. Next, we will formalise the above mentioned concepts introducing some examples.

DEFINITION 5.1 A DeLP-Server is a triple $\ll \mathbf{I}, \mathbf{O}, \mathcal{P} \gg$ where $\mathbf{I}$ is a DeLP-interpreter, $\mathbf{O}$ is a set of DeLP-program operators and $\mathcal{P}$ is a DeLP-program.

Thus, a DeLP-Server $\ll \mathbf{I}, \mathbf{O}, \mathcal{P} \gg$ provides a DeLP-interpreter, the possibility of representing public knowledge in the form of a DeLP-program, and a set of operators that will handle the integration of the contextual information with the program stored $\mathcal{P}$.

Consider, for example, a DeLP-server for the real state application mentioned above. This server will have some public knowledge represented by the program $\mathcal{P}_{RS} = (\Pi_{RS}, \Delta_{RS})$ shown below where we will use the following shorthand notation: *av* (available), *br* (bedroom), *conv_br* (convenient number of bedrooms). The set Π_{RS} contains facts representing information about the name, number of bedrooms, location and price of the properties available for renting. For instance, *av(p1,br(1),downtown,2000)* represents that the property $p1$ is available, it has 1 bedroom, is located downtown, and its price is 2000.

$$\Pi_{RS} = \left\{ \begin{array}{ll} av(p1,br(1),downtown,2000) & av(p2,br(2),suburbs,4000) \\ av(p3,br(3),suburbs,5000) & av(p4,br(3),downtown,15000) \\ av(p5,br(2),suburbs,10000) & av(p6,br(1),downtown,7000) \end{array} \right\}$$

$$\Delta_{RS} = \left\{ \begin{array}{l} suited(Property) \prec av(Property,br(B),\_,\_),conv\_br(B) \\ \sim suited(Property) \prec expensive(Property) \\ expensive(Property) \prec av(Property,\_,\_,Price),afford(A),Price > A \\ conv\_br(X) \prec has\_children,\ X \geq 2 \\ conv\_br(X) \prec couple,\ X \geq 1 \\ conv\_br(X) \prec single,\ X \geq 1 \\ has\_children \prec grown\_children \\ \sim conv\_br(2) \prec grown\_children \\ \sim conv\_br(1) \prec work\_at\_home \end{array} \right\}$$

These defeasible rules should interpreted as follows. A convenient number of bedrooms in any available property provide reasons for considering it as a suited property; observe that the reasons for and against a convenient number of bedrooms depends on the private information a client might send as a context. Properties are not considered suited if they are expensive; to determine if a property is expensive for a client, the amount the client can afford to expend has to be provided in the contextual query.

In the previous section we explained how a DeLP-interpreter takes a DeLP-program, a DeLP-query (a literal), and returns one of the four DeLP possible answers: YES, NO, UNDECIDED, and UNKNOWN. To put this more clearly, let $\mathbb{D}$ be the domain of all possible DeLP-programs, and $\mathbb{L}$ is the domain of DeLP-queries, then a DeLP-interpreter can be seen as a function

$$\mathbf{I} : \mathbb{D} \times \mathbb{L} \longrightarrow \{\text{YES, NO, UNDECIDED, UNKNOWN}\}$$

Then DeLP-program operator $o \in \mathbf{O}$, is a binary operator that will take two DeLP-programs $\mathcal{P}_1$ and $\mathcal{P}_2$, and it will return a new DeLP-program; this DeLP-program is the result of integrating $\mathcal{P}_1$ and $\mathcal{P}_2$ according to the definition of the operator o. Therefore, formally a DeLP-program operator can be seen as a function

$$o : \mathbb{D} \times \mathbb{D} \longrightarrow \mathbb{D}$$

A contextual query to a DeLP-Server involves two elements: a particular DeLP-query Q to be answered, and contextual information Co to be used together with the program stored in the

DeLP-Server to answer Q. This contextual information consists of both private knowledge in the form of a DeLP-program to be consider by the interpreter and appropriate operators that state how to integrate the private information with the public program stored at the server. Formally:

DEFINITION 5.2 A contextual query for a DeLP-Server $\ll \mathbf{I}, \mathbf{O}, \mathcal{P} \gg$ is a pair (Co,Q) where Q is a DeLP-query, and the context Co is a sequence $[(\{\mathcal{P}_1\}, o_1), \ldots, (\{\mathcal{P}_n\}, o_n)]$, where $\mathcal{P}_i$ is a DeLP-program and $o_i \in \mathbf{O}$, $1 \leq i \leq n$.

In García et al. (2007) and Tucat et al. (2009) concrete examples of program operators were given. These operators are modular in the sense that each particular server can have a different set of operators that can be designed specifically for the application domain in use. Recall that in DeLP the set Π has to be non-contradictory; therefore, a DeLP-program operator must return a program where Π is non-contradictory. Below, we include, as an example, three simple and basic operators denoted by $\oplus$, $\otimes$, and $\ominus$, that will be used throughout the examples below.

To simplify the presentation, these three operator will be defined for a restricted DeLP-program (Π, Δ) where Π can have facts but no strict rules. We introduce auxiliary notation to use in their definition. Let x be a fact, we denote with $\overline{x}$ the complement of x with respect to strong negation; that is $\overline{a} = {\sim}a$ and $\overline{{\sim}a} = a$. If X is a set of facts, the complement of X is defined as $\overline{X} = \{\overline{x} \mid x \in X\}$. Previously, we have mentioned that a DeLP-program operator can be seen as a function $o : \mathbb{D} \times \mathbb{D} \mapsto \mathbb{D}$; with that in mind we define the operators $\oplus$, $\otimes$, and $\ominus$ as follows:

$$(\Pi, \Delta) \oplus (\Pi_a, \Delta_a) = ((\Pi \setminus \overline{\Pi_a}) \cup \Pi_a, \Delta \cup \Delta_a)$$

$$(\Pi, \Delta) \otimes (\Pi_a, \Delta_a) = (\Pi \cup (\Pi_a \setminus \overline{\Pi}), \Delta \cup \Delta_a)$$

$$(\Pi, \Delta) \ominus (\Pi_a, \Delta_a) = (\Pi \setminus \Pi_a, \Delta \setminus \Delta_a)$$

The first two operators, $\oplus$ and $\otimes$, integrate the defeasible part of the program representing the private knowledge and the defeasible part of the common knowledge stored in the server. The last one, $\ominus$, will eliminate the defeasible rules stored in the server that are in common with the defeasible part of the private knowledge present in the contextual query.

The first operator, denoted by $\oplus$, will integrate the private knowledge of an agent received as a DeLP-program (Π_a, Δ_a) with the program stored in a server that contains (Π, Δ), giving priority to the information in Π_a. For instance, if the public knowledge in the server is $(\Pi, \Delta) = (\{a, b\}, \Delta)$ and the private information received in a contextual query is $(\Pi_a, \Delta_a) = (\{{\sim}a, c\}, \Delta_a)$, then

$$\begin{aligned}(\{a, b\}, \Delta) \oplus (\{{\sim}a, c\}, \Delta_a) &= ((\{a, b\} \setminus \overline{\{{\sim}a, c\}}) \cup \{{\sim}a, c\}, \Delta \cup \Delta_a) \\ &= ((\{a, b\} \setminus \{a, {\sim}c\}) \cup \{{\sim}a, c\}, \Delta \cup \Delta_a) = (\{b, {\sim}a, c\}, \Delta \cup \Delta_a)\end{aligned}$$

Note that a contradiction could arise if the private information would have been added directly because Π would have contained a and ${\sim}a$; by using this operator, since the received information has priority, the fact ${\sim}a$ will remain whereas a will be temporarily removed, and therefore ignored for answering the query.

In contrast, the operator denoted by $\otimes$ gives priority to the facts that are part of the DeLP-program (Π, Δ) stored in the server. For instance, if the public knowledge in the server is as before $(\Pi, \Delta) = (\{a, b\}, \Delta)$ and the received information is $(\Pi_a, \Delta_a) = (\{{\sim}a, c\}, \Delta_a)$, then

$$\begin{aligned}(\{a, b\}, \Delta) \otimes (\{{\sim}a, c\}, \Delta_a) &= (\{a, b\} \cup (\{{\sim}a, c\} \setminus \overline{\{a, b\}}), \Delta \cup \Delta_a) \\ &= (\{a, b\} \cup (\{{\sim}a, c\} \setminus \{{\sim}a, {\sim}b\}), \Delta \cup \Delta_a) = (\{a, b, c\}\Delta \cup \Delta_a)\end{aligned}$$

That is, since the stored information has priority, the fact ${\sim}a$ will remain whereas a will not be considered for answering the query.

Finally, the operator denoted by $\ominus$ was developed with the purpose of not considering for a particular query part of the public information maintained in the server. For instance, if the public knowledge in the server is $(\Pi, \Delta) = (\{a, b, p\}, \Delta)$ and the received information is $(\Pi_a, \Delta_a) = (\{a, b, \sim p\}, \Delta_a)$, then

$$(\{a, b, p\}, \Delta) \ominus (\{a, b, \sim p\}, \Delta_a) = (\{a, b, p\} \setminus \{a, b, \sim p\}, \Delta \setminus \Delta_a) = (\{p\}, \Delta \setminus \Delta_a)$$

To provide an answer for a particular contextual query (Co, Q), a DeLP-Server $\ll \mathbf{I}, \mathbf{O}, \mathcal{P} \gg$ has to consider first how the context Co is integrated with the stored program $\mathcal{P}$. Since the context can be a sequence of pairs $[(\{\mathcal{P}_1\}, o_1), \ldots, (\{\mathcal{P}_n\}, o_n)]$, then we define:

$$\mathcal{P} \diamond Co, \quad \text{as} \quad (o_n^{\mathcal{P}_n} \circ o_{n-1}^{\mathcal{P}_{n-1}} \circ \ldots \circ o_2^{\mathcal{P}_2} \circ o_1^{\mathcal{P}_1})(\mathcal{P}),$$

where $o_i^{\mathcal{P}_j}(\mathcal{P}_k)$ is a simplification of $o_i(\mathcal{P}_j, \mathcal{P}_k)$. Recall that DeLP-interpreter can be seen as a function $\mathbf{I} : (\mathbb{D} \times \mathbb{L}) \mapsto \{\text{YES}, \text{NO}, \text{UNDECIDED}, \text{UNKNOWN}\}$.

DEFINITION 5.3 Let $\ll \mathbf{I}, \mathbf{O}, \mathcal{P} \gg$ be a DeLP-server and (Co, Q) a contextual query. Then, an answer for (Co, Q) is $\mathbf{I}((\mathcal{P} \diamond Co), Q)$.

It is important to note that a context is a sequence and it will be processed by the DeLP-server in the order it appears in that sequence. Hence, different answers could be obtained depending on the order of the pairs in the context. Consider a DeLP-server $\ll \mathbf{I}, \{\oplus, \otimes, \ominus\}, (\Pi_{RS}, \Delta_{RS}) \gg$ for the real state application already introduced. We include some contextual queries that clients could send to the server and show the answers that will be returned:

$([(\{single, afford(3000)\}, \oplus)], suited(p1))$
answer: YES
$([(\{single, afford(3000)\}, \oplus)], suited(p2))$
answer: NO
$([(\{single, afford(4000)\}, \oplus)], suited(p2))$
answer: YES
$([(\{grown\_children, afford(4000)\}, \oplus)], suited(p2))$
answer: NO
$([(\{grown\_children, afford(9000)\}, \oplus)], suited(p3))$
answer: YES
$([(\{grown\_children, afford(4000)\}, \oplus)], suited(p3))$
answer: NO
$([(\{grown\_children, afford(4000)\}, \oplus), (\{\sim conv\_br(2) \prec grown\_children\}, \ominus)], suited(p2))$
answer: YES

6. Explanations for answers

In this section we will discuss a direct application of DeLP-Servers to address an important topic in computational reasoning systems. The research included here has been reported in García, Rotstein, Chesñevar, and Simari (2009) and García et al. (2013), where more extensive presentations have been given.

Giving explanations regarding an answer coming from a computational system after a query is a question that has been addressed in several areas of Artificial Intelligence; quite naturally, the expert systems community Lacave and Diez (2004), Ye and Johnson (1995), and Guida and Zanella (1997) have explored possible solutions. Researchers in the area argumentation also have been concerned with providing their views (Belanger, 2007; Moulin, Irandoust, Bélanger, & Desbordes, 2002; Walton, 2004); for instance, Walton (2004) has studied arguments and explanations from a philosophical point of view. He considers that the purpose of an argument is to get the hearer to come to accept something that is doubtful or unsettled, whereas the purpose of an explanation is to get the hearer to understand something that he already accepts as a fact.

In our approach, explanations aim to transfer the understanding of how the warrant status of a particular argument was obtained from a given argumentation framework. Thus, in our framework, an explanation (called δ-Explanation) will consist of a structure that reflects the analysis carried out in order to obtain such status, and it will contain those arguments and counterarguments that are considered in analysis leading to its warrant status.

In Lacave and Diez (2004), a review about explanations in heuristic expert systems is given and they offer the following definition:

> . . . explaining consists in *exposing something* in such a way that it is *understandable* for the receiver of the explanation – so that he/she improves his/her knowledge about the object of the explanation – and *satisfactory* in that it meets the receiver's expectations.

Using the language in the fragment above to describe our approach, we will *explain* the warrant status of a claim by *exposing* the whole set of dialectical trees that was generated to study the warrant status of arguments related to the claim; this information is *understandable* from the receiver's point-of-view because all the arguments considered, their statuses (i.e. defeated/undefeated), and their interrelations are explicitly shown, and this information will be *satisfactory* for the receiver because it contains all the elements at stake in the dialectical analysis that supports the answer.

Example 6.1 Consider for instance the following scenario (García et al., 2013). Bob is an opera fan and there is an opera show tonight. Besides, today is Bob's birthday and he usually gets together with friends. Bob usually goes to the opera house with friends. However, today Bob's best friend is coming with her baby because of Bob's birthday, and is not a good idea to go to the opera with a baby. The DeLP-program $(\Pi_{6.1}, \Delta_{6.1})$ represents the scenario just described. In this program, *go*, *show_tonight*, *birthday*, *baby* and *friends* stand for 'go to the opera show', 'there is an opera show tonight', 'today is Bob's birthday', 'a friend is coming with her baby', and 'get together with friends', respectively.

$$\Pi_{6.1} = \{show\_tonight, birthday, baby\}$$

$$\Delta_{6.1} = \left\{ \begin{array}{ll} go \prec showTonight & go \prec showTonight, friends \\ friends \prec birthday & \sim go \prec showTonight, friends, baby \\ \sim go \prec friends & \end{array} \right\}$$

This program contains three facts and five defeasible rules that represent tentative information. The first defeasible rule states that if there is an opera show tonight then there is a good reason to go to the opera house. The second defeasible rule represents that in his birthday Bob usually gets together with friends. The third one states that if friends are coming, Bob would probably stay at home. However, the fourth rule establishes that being with friends and the fact that there is an opera show that night will give a good reason to go to the opera. The fifth rule states that this situation changes if a friend is coming with a baby.

From program $(\Pi_{6.1}, \Delta_{6.1})$ the following arguments can be obtained:

$$\begin{aligned}
\langle \mathcal{O}_1, go\rangle &= \langle \{go \,\text{—}\!\!< showTonight\}, go\rangle\\
\langle \mathcal{O}_2, {\sim}go\rangle &= \langle \{({\sim}go \,\text{—}\!\!< friends), (friends \,\text{—}\!\!< birthday)\}, {\sim}go\rangle\\
\langle \mathcal{O}_3, go\rangle &= \langle \{(go \,\text{—}\!\!< showTonight, friends), (friends \,\text{—}\!\!< birthday)\}, go\rangle\\
\langle \mathcal{O}_4, {\sim}go\rangle &= \langle \{({\sim}go \,\text{—}\!\!< showTonight, friends, baby), (friends \,\text{—}\!\!< birthday)\}, {\sim}go\rangle
\end{aligned}$$

Let's assume the following priorities over rules:

$$({\sim}go \,\text{—}\!\!< friends) <_R (go \,\text{—}\!\!< showTonight, friends)$$

$$(go \,\text{—}\!\!< showTonight,\ friends) <_R ({\sim}go \,\text{—}\!\!< showTonight,\ friends, baby)$$

From this program and the associated priorities, the argument $\langle \mathcal{O}_2, {\sim}go\rangle$ represents a blocking defeater for $\langle \mathcal{O}_1, go\rangle$ and viceversa. The argument $\langle \mathcal{O}_3, go\rangle$ is a proper defeater for $\langle \mathcal{O}_2, {\sim}go\rangle$, due to the priority of the rules. The argument $\langle \mathcal{O}_4, {\sim}go\rangle$ is a proper defeater for both $\langle \mathcal{O}_3, go\rangle$ and $\langle \mathcal{O}_1, go\rangle$.

From the program $(\Pi_{6.1}, \Delta_{6.1})$ the answer corresponding to the query *go* is NO, and the answer for the query ${\sim}go$ is YES. As discussed previously, to obtain an answer for a DeLP-query Q, the interpreter goes through a dialectical process for warranting Q involving the construction and evaluation of the arguments that either support or interfere with the query under analysis. The generated arguments are affected by the defeat relation and they are organised in dialectical trees. Observe that given a DeLP-query Q, there might exist different arguments that support Q, and each argument will generate a separate dialectical tree. Therefore, as we will show below, the returned answer for Q will be only 'the tip of the iceberg' of a set of several dialectical trees that have been explored to support that answer; thus, to *understand* why a DeLP-query obtains a particular answer, it is essential to consider which arguments have been generated and what are the relations between them.

Since an explanation has to further the understanding of the warrant status of a DeLP-query Q, our approach to giving an explanation will consist of a structure that includes those marked dialectical trees that are considered for establishing such status. This will include both marked dialectical trees for arguments that support Q, and also marked dialectical trees for arguments that support $\overline{Q}$ (the complement of Q with respect to strong negation). For instance, returning to our Example 6.1, it is possible to build two arguments for *go* and two arguments that support the complement ${\sim}go$; therefore, the explanation should include these four trees. The following definition characterises two distinguished sets of marked dialectical trees that will be used to produce an explanation. Recall that we write $Mark(\mathcal{T}^*_{\langle \mathcal{A},L\rangle}) = U$ to denote that the root of $\mathcal{T}^*_{\langle \mathcal{A},L\rangle}$ is marked as "U" and we write $Mark(\mathcal{T}^*_{\langle \mathcal{A},L\rangle}) = D$ if the root of $\mathcal{T}^*_{\langle \mathcal{A},L\rangle}$ is marked as "D".

DEFINITION 6.2 Let $\mathcal{P}$ be a DeLP-program and Q a DeLP-query.

Let $\mathbb{T}^*(Q) = \{\langle \mathcal{A}_0, Q\rangle, \ldots, \langle \mathcal{A}_n, Q\rangle\}$ be the set of all arguments for Q from $\mathcal{P}$, and $\mathbb{T}^*(\overline{Q}) = \{\langle \mathcal{B}_0, \overline{Q}\rangle, \ldots, \langle \mathcal{B}_m, \overline{Q}\rangle\}$ the set of all arguments for $\overline{Q}$ from $\mathcal{P}$.

The sets $\mathbb{T}^*_U(Q) \subseteq \mathbb{T}^*(Q)$, and $\mathbb{T}^*_D(Q) \subseteq \mathbb{T}^*(Q) \cup \mathbb{T}^*(\overline{Q})$ are defined as follows:

$$\mathbb{T}^*_U(Q) = \{\mathcal{T}^*_{\langle \mathcal{A},L\rangle} \in \mathbb{T}^*(Q) \mid Mark(\mathcal{T}^*_{\langle \mathcal{A},L\rangle}) = U\},$$

$$\mathbb{T}^*_D(Q) = \{\mathcal{T}^*_{\langle \mathcal{A},L\rangle} \in (\mathbb{T}^*(Q) \cup \mathbb{T}^*(\overline{Q})) \mid Mark(\mathcal{T}^*_{\langle \mathcal{A},L\rangle}) = D\}.$$

Then, a dialectical explanation (δ-Explanation) for Q from $\mathcal{P}$, denoted by $\mathbb{E}_{\mathcal{P}}(Q)$, is the tuple $\mathbb{E}_{\mathcal{P}}(Q) = (\mathbb{T}^*_U(Q), \mathbb{T}^*_U(\overline{Q}), \mathbb{T}^*_D(Q))$.

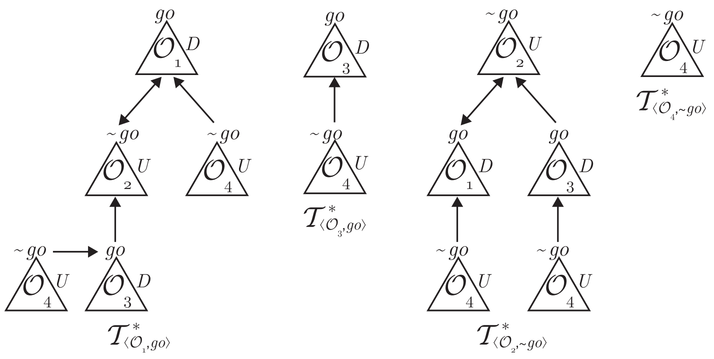
go
~go
D
U
$\mathcal{T}^*_{\langle \mathcal{O}_1, go \rangle}$
$\mathcal{T}^*_{\langle \mathcal{O}_3, go \rangle}$
$\mathcal{T}^*_{\langle \mathcal{O}_2, \sim go \rangle}$
$\mathcal{T}^*_{\langle \mathcal{O}_4, \sim go \rangle}$

Figure 5. Dialectical trees in the δ-Explanation for '$\sim go$'.

As stated above, the purpose of an explanation is to transfer the understanding of how the warrant status of a particular argument is obtained from a given argumentation framework. Consequently, an explanation will consist of a structure that reflects the analysis carried out in order to obtain such status, and it will contain those arguments and counterarguments that are considered in this analysis, showing explicitly how they relate to one another. That is, a δ-Explanation $\mathbb{E}_{\mathcal{P}}(Q)$ for a claim Q from a DeLP-program $\mathcal{P}$ is a triplet $(\mathbb{T}^*_U(Q), \mathbb{T}^*_U(\overline{Q}), \mathbb{T}^*_D(Q))$, where the first component is a (possibly empty) set of those marked dialectical trees from $\mathcal{P}$ that provide a warrant for Q. The second component of $\mathbb{E}_{\mathcal{P}}(Q)$is a (possible empty) set of marked dialectical trees that provide a warrant for $\overline{Q}$. Finally, the third component ($\mathbb{T}^*_D(Q)$) is a (possibly empty) set that contains those marked dialectical trees for Q or for $\overline{Q}$ that provide no warrant, i.e. their roots are marked D (defeated).

Consider the program $\mathcal{P}_{6.1} = (\Pi_{6.1}, \Delta_{6.1})$ of Example 6.1. The δ-Explanation for the query '$\sim go$' is:

$$\mathbb{E}_{\mathcal{P}_{6.1}}(\sim go) = (\{\mathcal{T}^*_{\langle \mathcal{A},L \rangle}\}, \emptyset, \{\mathcal{T}^*_{\langle \mathcal{A},L \rangle}, \mathcal{T}^*_{\langle \mathcal{A},L \rangle}, \mathcal{T}^*_{\langle \mathcal{A},L \rangle}\}).$$

Figure 5 shows all the marked trees included in this δ-Explanation.

6.1. *Schematic queries and generalised δ-Explanations*

A DeLP-schematic query to a DeLP-program is a literal that at least has one variable. It represents the set of DeLP-queries that unify with it only containing constants from the signature of the language of the program. Schematic queries introduce the possibility of asking questions that are more general than ground queries; for instance, consider the program introduced below in Example 6.3, the schematic query *flies*(X) represents the query *'what are the individuals for which there exists a warranting argument supporting that they fly?'* Next, we will extend the definition of δ-Explanation to include schematic queries and we will define the notion of answer for schematic queries.

Example 6.3 Consider the DeLP-program $(\Pi_{6.3}, \Delta_{6.3})$ where:

$$\Pi_{6.3} = \left\{ \begin{array}{l} \textit{bird(X)} \leftarrow \textit{chicken(X)} \\ \textit{chicken(little)} \\ \textit{chicken(tina)} \\ \textit{scared(tina)} \\ \textit{bird(rob)} \end{array} \right\} \qquad \Delta_{6.3} = \left\{ \begin{array}{l} \textit{flies(X)} \prec \textit{bird(X)} \\ \textit{flies(X)} \prec \textit{chicken(X),scared(X)} \\ \sim\textit{flies(X)} \prec \textit{chicken(X)} \end{array} \right\}$$

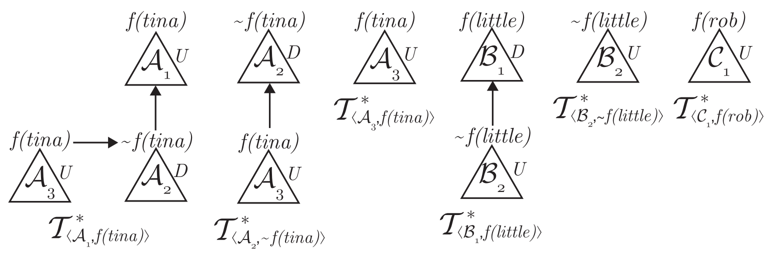
f(tina)
~f(tina)
f(tina)
f(little)
~f(little)
f(rob)
f(tina)
~f(tina)
f(tina)
~f(little)
$\mathcal{T}^*_{\langle \mathcal{A}_1, f(tina)\rangle}$
$\mathcal{T}^*_{\langle \mathcal{A}_2, \sim f(tina)\rangle}$
$\mathcal{T}^*_{\langle \mathcal{A}_3, f(tina)\rangle}$
$\mathcal{T}^*_{\langle \mathcal{B}_1, f(little)\rangle}$
$\mathcal{T}^*_{\langle \mathcal{B}_2, \sim f(little)\rangle}$
$\mathcal{T}^*_{\langle \mathcal{C}_1, f(rob)\rangle}$

Figure 6. Dialectical trees associated with the explanation for '*∼f(X)*'.

Observe that potentially, the variable X contained in the the schematic query *flies(X)* has an infinite number of terms that unify with that variable. However, all queries produced unifying with terms that are not in the signature of the language of this program; e.g. *flies(mac)* in Example 6.3, will produce the answer UNKNOWN and therefore the explanation will be empty. Thus, the set of instances of a schematic query that will be considered for generating a generalised δ-Explanation will refer only to those instances of DeLP-queries that contain constants from the program signature. For instance, in the DeLP-program of Example 6.3, the schematic query *flies(X)* will only refer to *flies(tina)*, *flies(rob)*, and *flies(little)*.

Schematic queries introduce the possibility of asking questions that are more general than ground queries. With this type of query we are not asking whether a certain piece of knowledge can be believed (warranted), instead we are asking if there exists an instance of that piece of knowledge (related to particular individuals) that can be warranted in the system. This approach could lead to a deeper kind of reasoning as we may pose a query, gather the warranted instances and continue the reasoning process with those individuals. As introduced next, an explanation for a schematic query will contain the explanation for the individual DeLP-queries it represents.

DEFINITION 6.4 Let $\mathcal{P}$ be a DeLP-program and Q a schematic query. Let $\{Q_1, \ldots, Q_z\}$ be the set of all ground instances of Q wrt the signature of $\mathcal{P}$. Let $\mathbb{E}_{\mathcal{P}}(Q_i)$ be the δ-Explanation for the DeLP-query Q_i $(1 \leq i \leq z)$ from program $\mathcal{P}$. Then, the generalised δ-Explanation for Q in $\mathcal{P}$ is

$$\mathbb{GE}_{\mathcal{P}}(Q) = \{\mathbb{E}_{\mathcal{P}}(Q_1), \ldots, \mathbb{E}_{\mathcal{P}}(Q_z)\}$$

It is interesting to observe that a δ-Explanation is a particular case of a Generalised δ-Explanation, where the set $\mathbb{GE}_{\mathcal{P}}(Q)$ is a singleton.

Consider again the DeLP-program $(\Pi_{6.3}, \Delta_{6.3})$, and suppose that we want to know if from this program it can be warranted that a certain individual does not fly. If we pose the query *∼flies(X)*, the answer is YES, because there is a warranted instance: *∼flies(little)*. The supporting argument is (where *'flies'* is abbreviated to *'f'*): $\langle \mathcal{B}_1, \sim f(little)\rangle = \langle\{\sim f(little) \prec chicken(little)\}, \sim f(little)\rangle$. The trees corresponding to the generalised explanation are shown in Figure 6. This explanation also shows that the other instance (*∼f(tina)*) is not warranted. Note that the answer for the schematic query *f(X)* is also YES, but with a different set of warranted instances: *f(tina)* and *f(rob)*. The supporting argument for instance '$X = tina$' was already discussed, and the undefeated argument for instance '$X = rob$' is: $\langle \mathcal{C}_1, f(rob)\rangle = \langle\{f(rob) \prec bird(rob)\}, f(rob)\rangle$. The generalised δ-Explanation for *f(X)* is:

$$\{(\{\mathcal{T}^*_{\langle\mathcal{A},L\rangle}, \mathcal{T}^*_{\langle\mathcal{A},L\rangle}\}, \emptyset, \{\mathcal{T}^*_{\langle\mathcal{A},L\rangle}\}), (\emptyset, \{\mathcal{T}^*_{\langle\mathcal{A},L\rangle}\}, \{\mathcal{T}^*_{\langle\mathcal{A},L\rangle}\}), (\{\mathcal{T}^*_{\langle\mathcal{A},L\rangle}\}, \emptyset, \emptyset)\}$$

Consider a schematic query $Q(X)$ and a DeLP-program $\mathcal{P}$. Suppose that $Q(X)$ represents the five (ground) DeLP-queries: $Q(a), Q(b), Q(c), Q(d)$ and $Q(e)$. As the reader may have noticed from the example above, it may happen that the answer for $Q(a)$ from $\mathcal{P}$ is YES, the answer for $Q(b)$ is NO, the answer for $Q(c)$ is YES, the answer for $Q(b)$ is UNDECIDED, and the answer for $Q(a)$ is NO. Next, we define the answer for a schematic query taking into consideration the individual answers for each ground instance.

Let $\mathcal{P}$ be a DeLP-program and Q a schematic query. Let $\{Q_1, \ldots, Q_n\}$ be all the ground instances of Q with respect to the signature of $\mathcal{P}$. The answer for the schematic query Q is:

- YES, if there exists an instance $Q_i \in \{Q_1, \ldots, Q_n\}$ such that the answer for the DeLP-query Q_i is YES (i.e. $\mathbb{T}^*_U(Q_i) \sim t = \emptyset$).
- NO, if for every instance $Q_i \in \{Q_1, \ldots, Q_n\}$, the answer for Q_i is NO, (i.e. $\mathbb{T}^*_U(\overline{Q_i}) \neq \emptyset$ *for all* Q_i).
- UNDECIDED, if there is no instance $Q_i \in \{Q_1, \ldots, Q_n\}$ such that the answer for the DeLP-query Q_i is YES, and there exists an instance $Q_k \in \{Q_1, \ldots, Q_n\}$ such that the answer for Q_k is UNDECIDED.
- UNKNOWN, if $\{Q_1, \ldots, Q_n\} = \emptyset$

For example, let's assume that from a particular program $\mathcal{P}$ all the ground instances for $p(X)$ with respect to the signature of $\mathcal{P}$ are $\{p(a), p(b), p(c)\}$ and for $t(X)$ are $\{t(a), t(b), t(c)\}$, for $k(X)$ are $\{k(a), k(b), k(c)\}$, and for $j(X)$ is the empty set. Consider that the following answers for ground queries are obtained:

$$p(a)\ \text{YES},\quad p(b)\ \text{NO},\quad p(c)\ \text{UNDECIDED},$$
$$t(a)\ \text{NO},\quad t(b)\ \text{NO},\quad t(c)\ \text{UNDECIDED},$$
$$k(a)\ \text{NO},\quad k(b)\ \text{NO},\quad k(c)\ \text{NO}.$$

The answer for $p(X)$ is YES because there is an instance ($p(a)$) for which the answer is YES. The answer for $t(X)$ is UNDECIDED because there is no warranted instance, and there is one instance ($t(c)$) for which the answer is UNDECIDED. The answer for $k(X)$ is NO because all the instances return NO. The answer for $j(X)$ is UNKNOWN.

Finally, note that from a DeLP programmer point-of-view, δ-Explanations provide a global idea of the interactions among arguments within the context of a query. This is an essential debugging tool while programming: whenever unexpected behaviour arises, the programmer can check the given explanations to detect errors.

7. Extensions and variations on DeLP

Several extensions of DeLP have been proposed and in this section we will briefly sketch a few of these developments. The following are some of these research lines that are more related to applications.

Possibilistic Defeasible Logic Programming (P-DeLP) (Alsinet, Chesñevar, Godo, Sandri, & Simari, 2008; Chesñevar, Simari, Alsinet, & Godo, 2004), is an extension of DeLP that allows the treatment of possibilistic uncertainty and fuzzy knowledge at object-language level in a logic programming language. In P-DeLP, knowledge representation features are formalised based on a possibilistic logic based on the Horn-rule fragment of Gödel fuzzy logic (PGL). Formulas in PGL are built over fuzzy propositional variables and the certainty degree of formulas is expressed with a necessity measure. The proof method for PGL, in a logic programming setting, is based on a complete calculus for determining the maximum degree of possibilistic entailment of a fuzzy goal. In a multiagent context, it is possible to use P-DeLP to encode the agents' knowledge about the world, applying the argument and warrant computing procedure to obtain their inferences. In particular, we have also a number of argument-based consequence operators which allow us to model different aspects of the reasoning abilities in an intelligent agent which have been formalised and studied; the computational problem related to how answers to P-DeLP queries can be sped up by pruning the associated search space have been considered.

The representation of strength and time provide useful elements in many argumentation applications. The Strength and Time DeLP, ST-DeLP, is an instantiation of the abstract framework E-TAF (Budán, Gómez Lucero, Chesñevar, & Simari, 2012) based on DeLP that incorporates the representation of temporal availability and strength factors varying over time associated with the elements of the language of DeLP. It also specifies how arguments are built, and how the availability and strength of arguments are obtained from the corresponding information attached to the elements from which they are built. After determining the availability of attacks by comparing strength of conflicting arguments over time, E-TAF definitions are applied to establish temporal acceptability of arguments. Thus, the main contribution of this development is the integration of time and strength in the context of argumentation systems. Other lines of work related to this can be found in Pardo and Godo (2011).

In García, García, and Simari (2008) an argumentation-based planning formalism has been introduced. This system can be used by an agent to construct plans using partial order planning technics, In such a manner that the traditional POP algorithm is extended to consider arguments as planning steps. When actions and arguments are combined to construct plans, new types of interferences appear ant it is necessary to extend the standard notion of threat to consider: action-action, action-argument, and argument-argument threats. The resulting algorithm is called APOP, and extends the traditional POP algorithm to consider actions and arguments as planning steps and resolve the new types of threats using new methods.

A framework for ontology integration of DL ontologies based on DeLP was introduced in Gómez, Chesñevar, and Simari (2013). In this work, the limitations for reasoning in the presence of inconsistent ontologies that Description Logic (DL) ontologies suffer are addressed using defeasible argumentation to model different DL reasoning capabilities when handling inconsistent ontologies, resulting in so-called d-ontologies. ONTOarg, provides a decision support framework for performing local-as-view integration of possibly inconsistent and incomplete ontologies in terms of DeLP. This ontology integration system is able to handle inconsistent DL-based ontologies by performing a dialectical analysis in order to determine the membership of individuals to concepts.

Another recent line of research deals with the problem of massively built arguments from premises obtained from relational databases and compute warrant (Deagustini et al., 2013). In this setting, different databases can be updated by external, independent applications, leading to changes in the spectrum of available arguments. Algorithms for integrating a database management system with an argument-based inference engine have been designed and tested. The empirical results and running-time analysis associated with the approach show how, taking advantage of modern DBMS technologies, it is possible to efficiently implement processes that requiere massive argumentation. This provides an interesting possibility for developing new architectures for knowledge-based applications, such as Decision Support Systems and Recommender Systems, that could efficiently use argumentation as the underlying inference model.

Note

1. We will use the term *knowledge base* and *defeasible logic program* interchangeably when no confusion may arise.

References

Alferes, J.J., Pereira, L.M., & Przymusinski, T.C. (1996). Strong and explicit negation in nonmonotonic reasoning and logic programming. *Lecture Notes in Computer Science, 1126*, 143–163.

Alsinet, T., Chesñevar, C.I., Godo, L., Sandri, S., & Simari, G.R. (2008). Formalizing argumentative reasoning in a possibilistic logic programming setting with fuzzy unification. *International Journal of Approximate Reasoning, 48*, 711–729.

Baral, C., & Gelfond, M. (1994). Logic programming and knowledge representation. *Journal of Logical Programming, 19/20*, 73–148.

Belanger, M. (2007). Exploitation of argumentation models for mission analysis. In T. Roth-Berghofer, S. Schulz, D. Bahls & D.B. Leake (Eds.), *Explanation-aware computing, papers from the 2007 AAAI workshop* (pp. 64–67). Vol. WS-07-06 of AAAI Tech. Report. Vancouver, Canada: AAAI Press.

Bench-Capon, T.J.M., & Dunne, P.E. (2007). Argumentation in artificial intelligence. *Artificial Intelligence, 171*, 619–641.

Besnard, P., & Hunter, A. (2001). A logic-based theory of deductive arguments. *Artificial Intelligence, 128*, 203–235.

Besnard, P., & Hunter, A. (2008). *Elements of argumentation*. Cambridge, MA: MIT Press.

Bondarenko, A., Dung, P.M., Kowalski, R.A., & Toni, F. (1997). An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence, 93*, 63–101.

Bondarenko, A., Toni, F., & Kowalski, R.A. (1993). An assumption-based framework for non-monotonic reasoning. In *Logic Programming and Nonmonotonic Reasoning, Lisbon, Portugal* (pp. 171–189), Boston, MA: The MIT Press.

Budán, M.C., Gómez Lucero, M.J., Chesñevar, C.I., & Simari, G.R. (2012). Modelling time and reliability in structured argumentation frameworks. In G. Brewka, T. Eiter & S.A. McIlraith (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, Rome, Italy, 2012*. Rome, Italy, AAAI Press (Menlo Park, CA), pp. 578–582.

Chesñevar, C.I., Maguitman, A.G., & Loui, R.P. (2000). Logical models of argument. *ACM Computing Surveys, 32*, 337–383.

Chesñevar, C.I., & Simari, G.R. (2007). A lattice-based approach to computing warranted beliefs in skeptical argumentation frameworks. In M.M. Veloso (Ed.), *Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India*, January (pp. 280–285), AAAI Press (Menlo Park, CA), Hyderabad, India.

Chesñevar, C.I., Simari, G.R., Alsinet, T., & Godo, L. (2004). A logic programming framework for possibilistic argumentation with vague knowledge. In D.M. Chickering & J.Y. Halpern (Eds.), *UAI* (pp. 76–84). Banff, Canada: AUAI Press.

Deagustini, C.A.D., Fulladoza Dalibón, S.E., Gottifredi, S., Falappa, M.A., Chesñevar, C.I., & Simari, G.R. (2013). Relational databases as a massive information source for defeasible argumentation. *Knowledge-Based Systems* (online).

Dung, P.M. (1993). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming. In R. Bajcsy (Ed.), *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France* (pp. 852–859), Morgan Kauffman (Burlington, MA), Chambéry, France.

Dung, P.M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence, 77*, 321–357.

Ferretti, E., Errecalde, M., García, A.J., & Simari, G.R. (2008). Decision rules and arguments in defeasible decision making. In P. Besnard, S. Doutre & A. Hunter (Eds.), *Proc. 2nd Int. Conference on Computational Models of Arguments (COMMA)* (pp. 171–182), Vol. 172 of Frontiers in Artificial Intelligence and Applications. Toulouse, France: IOS Press.

García, A.J. (2000). *Defeasible logic programming: Definition, operational semantics and parallelism* (Ph.D. thesis). Computer Science and Engineering Department, Universidad Nacional del Sur, Bahía Blanca, Argentina.

García, A.J., Chesñevar, C.I., Rotstein, N.D., & Simari, G.R. (2013). Formalizing dialectical explanation support for argument-based reasoning in knowledge-based systems. *Expert Systems with Applications, 40*, 3233–3247.

García, D.R., García, A.J., & Simari, G.R. (2008). Defeasible reasoning and partial order planning. In S. Hartmann & G. Kern-Isberner (Eds.), *FoIKS* (pp. 311–328). Vol. 4932 of Lecture Notes in Computer Science. Pisa, Italy: Springer.

García, A.J., Rotstein, N.D., Chesñevar, C.I., & Simari, G.R. (2009). Explaining why something is warranted in defeasible logic programming. In *ExaCt 2009* (pp. 25–36). Pasadena, CA: IJCAI.

García, A.J., Rotstein, N.D., Tucat, M., & Simari, G.R. (2007). An argumentative reasoning service for deliberative agents. In Z. Zhang & J.H. Siekmann (Eds.), *Proceedings of the Second International Conference on Knowledge Science, Engineering and Management, KSEM 2007, November 28–30* (pp. 128–139). Vol. 4798 of Lecture Notes in Computer Science. Melbourne: Springer.

García, A.J., & Simari, G.R. (2004). Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming, 4*, 95–138.

Gómez, S.A., Chesñevar, C.I., & Simari, G.R. (2013). ONTOarg: A decision support framework for ontology integration based on argumentation. *Expert Systems with Applications, 40*, 1858–1870.

Guida, G., & Zanella, M. (1997). Bridging the gap between users and complex decision support systems: The role of justification. In *ICECCS '97: Proc. 3rd IEEE Int. Conf. on Eng. of Complex Computer Systems* (pp. 229–238). Washington.

Lacave, C., & Diez, F.J. (2004). A review of explanation methods for heuristic expert systems. *Knowledge Engineering Review, 19*, 133–146.

Lifschitz, V. (1996). Foundations of logic programs. In G. Brewka (Ed.), *Principles of knowledge representation* (pp. 69–128). Stanford: CSLI Pub.

Lin, F., & Shoham, Y. (1989). Argument systems: A uniform basis for nonmonotonic reasoning. In R.J. Brachman, H.J. Levesque & R. Reiter (Eds.), *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)* (pp. 245–255). Toronto, Canada: Morgan Kaufmann.

Loui, R.P. (1987). Defeat among arguments: A system of defeasible inference. *Computational Intelligence, 3*, 100–106.

Martínez, M.V., García, A.J., & Simari, G.R. (2012). On the use of presumptions in structured defeasible reasoning. In B. Verheij, S. Szeider & S. Woltran (Eds.), *Computational Models of Argument, Proceedings of COMMA 2012* (pp. 185–196). Vol. 245 of Frontiers in Artificial Intelligence and Applications. Vienna, Austria: IOS Press.

McCarthy, J. (1980). Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence, 13*, 27–39.

McDermott, D.V., & Doyle, J. (1980). Non-monotonic logic I. *Artificial Intelligence, 13*, 41–72.

Modgil, S., & Prakken, H. (2013). A general account of argumentation with preferences. *Artificial Intelligence, 195*, 361–397.

Moore, R.C. (1984). Possible-world semantics for autoepistemic logic. In *Workshop on Nonmotononic Reasoning* (pp. 344–354), AAAI Press (Menlo Park, CA), New Paltz, NY.

Moulin, B., Irandoust, H., Bélanger, M., & Desbordes, G. (2002). Explanation and argumentation capabilities: Towards the creation of more persuasive agents. *Artificial Intelligence Review, 17*, 169–222.

Nute, D. (1987). Defeasible reasoning. In *Proceedings of the 20th Hawaii International Conference on System Science*, Hawaii, USA (pp. 470–477).

Nute, D. (1988). Defeasible reasoning: A philosophical analysis in PROLOG. In J.H. Fetzer (Ed.), *Aspects of artificial intelligence* (pp. 251–288). Dordrecht, The Netherlands: Kluwer Academic Pub.

Pardo, P., & Godo, L. (2011). t-DeLP: A temporal extension of the defeasible logic programming argumentative framework. In S. Benferhat & J. Grant (Eds.), *SUM 2011 Proceedings of the 5th International Conference on Scalable Uncertainty Management* (pp. 489–503). Vol. 6929 of Lecture Notes in Computer Science. Dayton, Ohio: Springer.

Pollock, J.L. (1987). Defeasible reasoning. *Cognitive Science, 11*, 481–518.

Pollock, J.L. (1996). A general-purpose defeasible reasoner. *Journal of Applied Non-Classical Logics, 6*, 89–113.

Prakken, H. (2010). An abstract framework for argumentation with structured arguments. *Argument & Computation, 1*, 93–124.

Prakken, H., & Vreeswijk, G. (2002). Logics for defeasible argumentation. In D. Gabbay & F. Guenthner (Eds.), *Handbook of philosophical logic* (pp. 218–319). Dordrecht, The Netherlands: Kluwer Academic Pub.

Rahwan, I., & Simari, G.R. (2009). *Argumentation in artificial intelligence*. Heidelberg, Germany: Springer.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intellence, 13*, 81–132.

Simari, G.R. (1989). *A mathematical treatment of defeasible reasoning and its implementation*. Washington University, Department of Computer Science (Saint Louis, MO: EE.UU.).

Simari, G.R., Chesñevar, C.I., & García, A.J. (1994a). Focusing inference in defeasible argumentation. In *IV Iberoamerican Conference on Artificial Intelligence*, October, IBERAMIA'94, Caracas, Venezuela.

Simari, G.R., Chesñevar, C.I., & García, A.J. (1994b). The role of dialectics in defeasible argumentation. In *XIV International Conference of the Chilenean Computer Science Society*, November, Sociedad Chilena de Ciencias de la Computacion, Concepcion, Chile.

Simari, G.R., & Loui, R.P. (1992). A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence, 53*, 125–157.

Tucat, M., Garcia, A.J., & Simari, G.R. (2009). Using defeasible logic programming with contextual queries for developing recommender servers. In *AAAI Fall Symposium Series*, AAAI Press (Menlo Park, CA), Arlington, VA.

Walton, D. (2004). A new dialectical theory of explanation. *Philosophical Explorations, 7*, 71–89.

Ye, L.R., & Johnson, P.E. (1995). The impact of explanation facilities on user acceptance of expert systems advice. *MIS Quarterly, 19*, 157–172.