A Redefinition of Arguments in Defeasible Logic Programming

Ignacio Viglizzo and Fernando Tohmé and Guillermo Simari
Universidad Nacional del Sur, Bahía Blanca, Argentina
{viglizzo,fthome}@criba.edu.ar, grs@cs.uns.edu.ar

Abstract
Defeasible Logic Programming (DeLP) is a formalism that extends declarative programming to capture defeasible reasoning. Its inference mechanism, upon a query on a literal in a program, answers by indicating whether or not it is warranted in an argumentation process. While the properties of DeLP are well known, some of its basic elements can be redefined in order to shed light on some of the subtleties of the warrant process. We will discuss these alternative definitions and the cases in which they provide a better performance.

Introduction
The inference mechanism known as Defeasible Logic Programming (from now on DeLP) has been studied for some time now, and applied to many fields. Many extensions have been considered. This paper intends to take a step back to give a rigorous look at its formal foundations and suggest some possible improvements.

We present here a redefinition of the basic elements of DeLP (García and Simari 2004) in order to put this framework in line with recent developments in defeasible reasoning. This version, by considering alternative characterizations of those elements, provides a enhanced formal approach to DeLP.

In DeLP the goal is to determine whether literals in a logic program including defeasible rules are warranted or not. It proceeds by constructing arguments for and against any given literal and drawing different argumentation lines (i.e. sequences of arguments, in which each one attacks the previous one). The whole family of argumentation lines for a literal defines a dialectical tree. A marking procedure, finally, determines if the literal becomes warranted or not.

The basic issues in DeLP involve the definition of argument, the condition of defeat among them as well as the admissibility conditions for the construction of argumentation lines. In the original version arguments regarded only defeasible rules, but this was a source for some problems in the characterization of defeat, as we noted in (Viglizzo, Tohmé, and Simari 2008). In the current version, these problems disappear, thanks to slight changes in the definitions. We have chosen to introduce first our new system and left the comparison with the existing one for the last section.

In the next section we present the motivations of the present work. In the following one we introduce the new definitions and then we compare our new proposal with the existing framework. The main point is that this new approach provides a more flexible formal model, able to handle some finer distinctions among arguments.

Preliminaries: A New Look at DeLP
A central topic in this paper is a new definition of argument in DeLP to include more structure. That will allow us to enhance the notion of warrant. Nevertheless, we will maintain the intuitions behind the original presentation (García and Simari 2004; García 2000; García and Simari 1999) of DeLP. The motivation of the present work can be found in some developments (Viglizzo, Tohmé, and Simari 2008; Falappa, Kern-Isberner, and Simari 2002; Stolzenburg et al. 2003; Chesnèvar et al. 2003; Chesnèvar and Simari 2001) that show that a finer, more structured, version of DeLP could enhance the possibilities of applying this framework in Multi-Agent Systems.

The general approach to reasoning in DeLP starts with a defeasible logic program and takes a dialectical view of the process of inference. In classical logic, proofs are undisputed. A well-formed formula, wff, $L$ is said to be a consequence of a set $S$ of wffs if and only if there exists a sequence $L_1, L_2, \ldots, L_n$ of wffs such that $L$ is $L_n$ and, for each $i$ either $L_i$ is an axiom or $L_{i+1}$ is a direct consequence by some rule of inference of some of the preceding wffs in the sequence. The sequence $L_1, L_2, \ldots, L_n$ is called a proof.

In DeLP, some of the rules are defeasible and can be challenged. A literal $L$ will be warranted if there exists a non-defeated argument supporting $L$. An argument $A$ for a literal $L$ is a minimal and consistent set of rules that allows to infer $L$. In order to establish whether the argument $A$ is a non-defeated argument, argument rebuttals or counter-arguments that could be defeaters for the original argument are considered and compared using some preference criterion, to decide if they are better than the attacked argument. Since counter-arguments are arguments, there may exist de-
The following program will denote \( P \Delta = \{ \} \) as written as: \( \Pi \) traditional notation as: \( A \overset{f}{\Rightarrow} \Delta \) \( \Pi \). Logic Program \( L \) Head ordered pair, denoted “\( L \) head”, a finite non-empty set of literals. A defeasible rule with \( l \in \) set of all literals in \( A \) has an atom, and whose second member, \( r \) is an literal, and whose second member, \( x \) is an atom \( l \in L \) and body \( \{ L_1, \ldots, L_n \} \) can also be written as: \( L_0 \leftarrow L_1, \ldots, L_n \) \( n > 0 \) or \( L_1 \land \ldots \land L_n \rightarrow L_0 \). 

Definition 4 (Defeasible Rule) A Defeasible Rule is an ordered pair, denoted “\( Head \leftarrow Body \)”, whose first member, \( Head \), is a literal, and whose second member, \( Body \), is a finite, non-empty set of literals. A defeasible rule with head \( L_0 \) and body \( \{ L_1, \ldots, L_n \} \) can also be written as: \( L_0 \leftarrow L_1, \ldots, L_n \) \( n > 0 \), or \( L_1 \land \ldots \land L_n \rightarrow L_0 \). 

Definition 5 (Defeasible Logic Program) A Defeasible Logic Program \( P \), abbreviated delp, is the disjoint union of three possibly infinite sets: a set of facts \( I_f \), a set of strict rules \( I_r \), and a set of defeasible rules \( \Delta \). When required, we will denote \( P \) as \( (I_f, I_r, \Delta) \), where \( I_f = I_f \cup I_r \). 

Example 1 The following program \( P \Delta = (I_f, \Delta) \), written in the traditional notation as: \( I_f = \{ r \leftarrow s; p; s \} \) \( \Delta = \{ q \leftarrow p; q \rightarrow p; r \} \) will be written as: \( I_f = \{ p, s \} \) \( I_r = \{ s \rightarrow r \} \) \( \Delta = \{ p \rightarrow q, p \land r \rightarrow q \} \) 

Definition 6 (Defeasible and Strict Derivation) Let \( P \) be a delp and \( L \) a ground literal. A derivation of \( L \) from \( P \), denoted \( P \vdash L \), consists of a finite sequence \( L_1, L_2, \ldots, L_n = L \) of ground literals, and each literal \( L_i \) is in the sequence because: 
(i) \( L_i \in \Pi_f \), or 
(ii) there exists a rule \( R_i \in \Pi_r \) with head \( L_i \) and body \( B_1, B_2, \ldots, B_n \) and every literal of the body is an element \( L_j \) of the sequence appearing before \( L_i \), \( j < i \). 
If all the rules used in the derivations come from the set \( \Pi_f \), then we say that the sequence is a strict derivation. In this case we will write \( P \vdash L \). If at least one of the rules comes from \( \Delta \), we will say it is a defeasible derivation. 

Definition 7 (Annotated Derivation) Let \( P \) be a delp and \( L \) a ground literal. An annotated derivation of \( L \) from \( P \), consists of a finite sequence of rules and facts \( [R_1, R_2, \ldots, R_n] \), where \( L \) is the fact \( R_n \), or the head of the rule \( R_n \) and if a rule \( R_i \) in the sequence, then its body \( B_1 \land \ldots \land B_k \), is such that for all \( B_j \), \( 1 \leq j \leq k \), \( B_j \) is a fact or it appears as the head \( L_m \), for some rule \( R_m \) with \( 1 \leq m < i \). 

Remark 1 Notice that facts could be added to the sequence at any point prior to their appearance in the body of a rule. Even so, we might endow the set of facts and rules with a total order so that there is a way to choose one of all the annotated defeasible derivations of a literal that contain the same rules, by using a lexicographical order on those sequences. 

Definition 8 (Annotated Sub-derivation) Let \( P \) be a delp and \( L \) a ground literal. Let \( D \) be an annotated derivation of \( L \) from \( P \). We will say that an annotated derivation \( D' \) from \( P \) is a sub-derivation of \( D \) if every member of \( D' \) is a also a member of \( D \). 

Example 2 From the program \( P \): 
\( I_f = \{ p, s \} \) 
\( I_r = \{ s \rightarrow r \} \) 
\( \Delta = \{ p \rightarrow q, p \land r \rightarrow q \} \), 
the next annotated derivation \( D \) for \( q \) can be constructed: 
\( [s, s \rightarrow r, p, p \land r \rightarrow q] \) and the annotated derivation \( D' = [s, s \rightarrow r] \) is a sub-derivation of \( D \). 

From the point of view of the analysis of the process of reasoning, interesting derivations must be concise. 

Definition 9 (Minimal Derivation) Let \( P \) be a delp and \( L \) a ground literal. Let \( D \) be an annotated derivation of \( L \) from \( P \), we will say that \( D \) is a minimal annotated derivation if there is no proper sub-derivation \( D' \) of \( D \) such that \( D' \) is also an annotated derivation of \( L \). 

Remark 2 Every derivation \( D \) for \( L \) contains at least a minimal sub-derivation \( D' \) for that literal \( L \).
As we will see, introducing unnecessary elements in a defeasible derivation weakens its ability to support its conclusion. For that reason, we will only consider minimal derivations.

There are other ways in which derivations can be minimal. Consider the following example:

**Example 3** Let \( \mathcal{P} \) be the program composed of \( \Pi_f = \{a, b\}; \Pi_r = \{a \land b \rightarrow c, b \rightarrow e, d \rightarrow g, g \rightarrow f\}; \Delta = \{c \leftarrow f, a \leftarrow d, e \leftarrow f\} \).

We have the following minimal derivations for the literal \( f \):

\[
[a, b, a \land b \rightarrow c, c \leftarrow f] \\
[a, a \leftarrow d, d \rightarrow g, g \rightarrow f] \\
[b, b \rightarrow e, e \leftarrow f]
\]

While the three of them are minimal in the sense defined above, we can observe that the first two are longer than the third one, and that the first one uses both facts \( a \) and \( b \), while the second one uses only \( a \).

One may also want to avoid using defeasible rules as much as possible. This adds yet another dimension to the study of these derivations.

We can also observe that all three are minimal sub-derivations for \( f \) of the annotated derivation:

\[
[a, b, a \land b \rightarrow c, c \leftarrow f, a \leftarrow d, d \rightarrow g, g \rightarrow f, b \rightarrow e, e \leftarrow f]
\]

One of the properties that a piece of reasoning should have is that a contradictory program is inconsistent, that is,\( C(\Pi_f, r, \emptyset) \) is not contradictory.

**Definition 12 (Consistent Derivation)** Let \( D \) be an annotated derivation. We will say that \( D \) is consistent if the set of all rules and facts used in \( D \) is consistent.

**Definition 13 (Brief)** We call a minimal, consistent annotated derivation a brief.

**Definition 14 (Argument)** Let \( \mathcal{P} \) be a delp, and let \( D \) be a brief for a literal \( L \) from \( \mathcal{P} \). We will say that the set \( A \) of facts and rules, strict and defeasible, contained in the derivation \( D \) is an argument for \( L \) derived from \( \mathcal{P} \).

So an argument for \( L \) constructed from \( \mathcal{P} \) is the set obtained from applying a forgetful operator \( \text{Arg} \) to a brief. If \( D \) is a brief we can write the argument obtained as \( A = \text{Arg}(D) \).

**Remark 4** From the definitions above, we notice that given an annotated derivation \( D \), constructed from a program \( \mathcal{P} \), the set of rules and facts involved in that derivation represents a subprogram of \( \mathcal{P} \), and therefore an argument for a literal \( L \) is a minimal, consistent subprogram \( \mathcal{A} \) of \( \mathcal{P} \) from which \( L \) can derived.

**Notation:** We will use the notation \( \langle A, L \rangle_{\mathcal{P}} \) to represent the fact that \( A \) is an argument for \( L \) constructed from \( \mathcal{P} \). Usually, when the reference to the defeasible logic program \( \mathcal{P} \) is clearly understood, we will simplify the notation for arguments as \( \langle A, L \rangle \). Unless it is necessary, we will not mention the defeasible program \( \mathcal{P} \). Given a defeasible logic program \( \mathcal{P} \), we will denote the set of all arguments \( \langle A, L \rangle \) that can be constructed from \( \mathcal{P} \) as \( \text{Arg}(\mathcal{P}) \).

**Remark 5** While we keep the notation above, we observe that the literal \( L \) is redundant as the next proposition shows.

**Proposition 1** Given an argument \( A \), there is a unique literal \( L \) so that the facts and rules in a set \( A \) can be listed as a brief for \( L \).

**Proof.** Assume that there are two such literals \( L_1 \) and \( L_2 \). Let \( \{R_1, R_2, \ldots, R_n\} \) be a brief for \( L_1 \). Assume that \( L_2 \) is the head of rule \( R_i \). If \( i < n \), then \( \{R_1, \ldots, R_i\} \) is a shorter brief for \( L_2 \), and this contradicts the minimality of \( A \). Therefore \( R_i = R_n \) and \( L_1 = L_2 \).

**Definition 15 (Sub-Argument)** Let \( \mathcal{P} \) be a delp, and let \( A \) be an argument for a literal \( L \) constructed from a brief \( D \) produced from \( \mathcal{P} \). If \( D \) is a brief for a literal \( L_1 \in C(A) \) and a sub-derivation of \( D \), then \( A_1 = \text{Arg}(D_1) \) is a sub-argument of \( A \). It is clear that \( A_1 \subseteq A \) and we will also use the notation \( \langle A_1, L_1 \rangle \subseteq \langle A, L \rangle \) to reflect this.

**Proposition 2** Let \( \mathcal{P} \) be a delp, and let \( A \) be an argument for a literal \( L \) constructed from a brief \( D \) produced from \( \mathcal{P} \). Then, \( L_1 \in C(A) \) if and only if there is a unique sub-argument \( \langle A_1, L_1 \rangle \) of \( \langle A, L \rangle \).
Definition 17 (Counter-argument) Let \( \lambda \) denote the unique sub-argument \( \langle A, \text{argues}, C \rangle \) of a pair \( \langle A, L \rangle \). Then \( \lambda \) is a sub-argument of \( \langle A, L \rangle \) if and only if \( (A, L) \) is a sub-argument of \( \lambda \) and \( \lambda \) is such that \( \lambda \) is a blocking term of \( \langle A, L \rangle \).

The literals \( \lambda \) is a blocking term of \( \langle A, L \rangle \) if \( \langle A, L \rangle \) and \( \lambda \) satisfy the following conditions:

1. \( \lambda \) is a blocking term of \( \langle A, L \rangle \).
2. \( \lambda \) is a blocking term of \( \langle A, L \rangle \).
3. \( \lambda \) is a blocking term of \( \langle A, L \rangle \).
4. \( \lambda \) is a blocking term of \( \langle A, L \rangle \).
5. \( \lambda \) is a blocking term of \( \langle A, L \rangle \).

We call \( \lambda \) a blocking term of \( \langle A, L \rangle \) if \( \langle A, L \rangle \) is a blocking term of \( \lambda \) and \( \lambda \) is such that \( \lambda \) is a blocking term of \( \langle A, L \rangle \).

Definition 18 (Defeaters) \( \langle A_1, L_1 \rangle \) is a proper defeater of \( \langle A_2, L_2 \rangle \) (at literal \( L \)) if \( \langle A_2, L_2 \rangle R \langle A_1, L_1 \rangle \) and the unique sub-argument \( \lambda \) of \( \langle A_2, L_2 \rangle \) is such that \( \langle A_2, L_2 \rangle \lambda \langle A_1, L_1 \rangle \).

The literals \( \lambda \) is a blocking defeater of \( \langle A_2, L_2 \rangle \) (at literal \( L \)) if \( \langle A_2, L_2 \rangle R \langle A_1, L_1 \rangle \) and the unique sub-argument \( \lambda \) of \( \langle A_2, L_2 \rangle \) is such that \( \langle A_2, L_2 \rangle \neq \langle A_1, L_1 \rangle \) and \( \langle A_1, L_1 \rangle \neq \langle A_2, L_2 \rangle \).

When the is the case we use the notation \( \langle A_2, L_2 \rangle \approx \langle A_1, L_1 \rangle \), but we must keep in mind this is not necessarily a symmetric relation.

We call defeaters of an argument to all proper and blocking defeaters. If \( \langle A_2, L_2 \rangle \) is a defeater of \( \langle A_1, L_1 \rangle \) we write \( \langle A_1, L_1 \rangle \leq \langle A_2, L_2 \rangle \).

For the sake of completeness we present next the definitions, taken from García and Simari, 2004 that conform the DeLP system, although we have not made significant changes on them.

Definition 19 (Argumentation Line) Let \( \mathcal{P} \) be a delp and \( \langle A_0, L_0 \rangle \) an argument obtained from \( \mathcal{P} \). An argumentation line for \( \langle A_0, L_0 \rangle \) is a sequence of arguments from \( \mathcal{P} \), denoted \( \Lambda = [\langle A_0, L_0 \rangle, \langle A_1, L_1 \rangle, \langle A_2, L_2 \rangle, \langle A_3, L_3 \rangle, \ldots] \), where each element of the sequence \( \langle A_i, L_i \rangle \), \( i > 0 \), is a defeater of its predecessor \( \langle A_{i-1}, L_{i-1} \rangle \).

Definition 20 (Supporting and Interfering Arguments) Let \( \Lambda = [\langle A_0, L_0 \rangle, \langle A_1, L_1 \rangle, \langle A_2, L_2 \rangle, \langle A_3, L_3 \rangle, \ldots] \) be an argumentation line. We define the set of supporting arguments as \( \mathcal{A}_S = \{ \langle A_0, L_0 \rangle, \langle A_2, L_2 \rangle, \ldots \} \), and the set of interfering arguments \( \mathcal{A}_I = \{ \langle A_1, L_1 \rangle, \langle A_3, L_3 \rangle, \ldots \} \).

Definition 21 (Concordance) Let \( \mathcal{P} \) be a delp. Two arguments \( \langle A_1, L_1 \rangle \) and \( \langle A_2, L_2 \rangle \) are concordant if and only if the set \( \mathcal{C}(\mathcal{P}) \) is non-contradictory. More generally, a set of arguments \( \{ \langle A_i, L_i \rangle \} \) is concordant if and only if \( \mathcal{C}(\mathcal{P}) \) is non-contradictory.

Definition 22 (Acceptable Argumentation Line) An argumentation line \( \Lambda = [\langle A_1, L_1 \rangle, \ldots, \langle A_n, L_n \rangle] \) is an acceptable argumentation line if and only if:

1. \( \Lambda \) is a finite sequence.
2. The sets \( \mathcal{A}_S \) and \( \mathcal{A}_I \) of supporting and interfering arguments are concordant.
3. No argument \( \langle A_k, L_k \rangle \) in \( \Lambda \) is a sub-argument of an argument \( \langle A_i, L_i \rangle \) appearing earlier in \( \Lambda \) (\( i < k \)).
4. For all \( i \) such that the argument \( \langle A_i, L_i \rangle \) is a blocking defeater for \( \langle A_{i-1}, L_{i-1} \rangle \), if \( \langle A_{i+1}, L_{i+1} \rangle \) exists, then it is a proper defeater for \( \langle A_i, L_i \rangle \).

Definition 23 (Dialectical Tree) A dialectical tree for an argument constructed from a program \( \mathcal{P} \) for \( \langle A, L \rangle \), denoted \( T(\mathcal{P}, \langle A, L \rangle) \), is formed taking all the acceptable argumentation lines that start with \( \langle A, L \rangle \): the line \( \langle A, L \rangle \) is the root of the tree and a line \( \langle A, L \rangle, \langle A_1, L_1 \rangle, \ldots, \langle A_n, L_n \rangle \) is the child of \( \langle A, L \rangle, \langle A_1, L_1 \rangle, \ldots, \langle A_n, L_n \rangle \) for all \( n \geq 1 \).

Procedure 1 (Marking of a Dialectical Tree) Let \( T(\mathcal{P}, \langle A, L \rangle) \) be a dialectical tree for \( \langle A, L \rangle \). The corresponding marked dialectical tree, denoted \( T(\mathcal{P}, \langle A, L \rangle) \), will be obtained marking every node in \( T(\mathcal{P}, \langle A, L \rangle) \) as follows:

1. All leaves in \( T(\mathcal{P}, \langle A, L \rangle) \) are marked as “U”s in \( T(\mathcal{P}, \langle A, L \rangle) \).
2. Let \( \Lambda \) be an inner node of \( T(\mathcal{P}, \langle A, L \rangle) \). Then \( \Lambda \) will be marked as “U” in \( T(\mathcal{P}, \langle A, L \rangle) \) if and only if every one of its children is marked as “D”. Otherwise, \( \Lambda \) will be marked as “D”.
Definition 24 (Warranted Literal) A literal $L$ is warranted if and only if for some argument $(A, L)$, the root of $T_{(A,L)}$ is marked as “$U$”. We will say that $A$ is a warrant for $L$.

The definition of warrant ensures the non-monotonicity of DLP. If a new rule is added to a program $P$, the dialectical tree for $(A, L)$ may change and consequently, instead of being marked “$U$”, its root may become marked as “$D$”.

Example 7 Let $P$ be the program: $\Pi_f = \{p\}, \Pi_r = \{p \rightarrow b\}$. Let $A = \{p, p \rightarrow b\}$, $B = \{p, p \rightarrow b, b \sim f\}$. It follows trivially that $f$ is warranted by $B$.

Now consider $P'$, in which $\Pi'_f = \Pi_f, \Pi'_r = \Pi_r$ while $\Pi' = \Pi \cup \{p \sim f\}$. Then, a new argument obtains, $\Delta' = \Delta \cup \{p \sim f\}$. If furthermore, $(B, f) \prec (\Delta', \sim f)$, then $(B, f) \prec (\Delta, \sim f)$ so in $P'$, $T'_{(B,f)}$ is such that $f$ is no longer warranted.

Comparison with the previous framework
In this section we want to compare the newly introduced system with the one originally presented in Simari and Loui, 1992 and García and Simari, 2004. In order to do this, we must distinguish the arguments as defined in Definition 14 of this paper from the original ones, which we’ll call $d$-arguments. Here the letter $d$ stands to emphasize that only the defeasible rules are regarded in this kind of arguments.

Definition 25 (d-arguments) Let $P = (\Pi, \Delta)$ be a delp. A $d$-argument is a pair $(A, L)$ such that there is a brief $D$ for $L$ and $A = \Delta \cap \text{Arg}(D)$.

The definition above is not the original one but one can readily see it is equivalent to it, and uses the machinery introduced in this paper.

Now we can be recover $d$-arguments from our arguments.

Definition 26 (Equi-defeasibility) Let $P = (\Pi, \Delta)$ be a delp and $\text{Arg}(P)$ the corresponding set of arguments. Let $(A_1, L_1), (A_2, L_2) \in \text{Arg}(P)$. We will say that $A_1$ and $A_2$ are equi-defeasible if only if $A_1 \cap \Delta = A_2 \cap \Delta$. We will denote this relation as $A_1 \equiv d A_2$.

That is, $A_1$ and $A_2$ are equi-defeasible if they coincide on the set of defeasible rules they use. It is easy to see that equi-defeasibility is an equivalence relation.

There is a correspondence between arguments $A$ supporting a literal $L$ and $d$-arguments $(A, L)$, where $[A]$ denotes the equivalence class of $A$ under the equi-defeasibility relation. We will identify the class of an argument $A, [A]$ with the set $A \cap \Delta$. Together with d-arguments, we have the corresponding definitions of $d$-sub-argument, $d$-attack and $d$-defeating relation:

Definition 27 Let $d\text{Arg}(P)$ be the set of all $d$-arguments of a program $P$. The $d$-argument $(A_1, h)$ is a $d$-sub-argument of $(A_2, h')$ iff $A_1 \subseteq A_2$.

Definition 28 For each literal $h$ we define the binary relation $R_d^h$ on $d\text{Arg}(P)$ by $(A_1, h_1) R_d^h (A_2, h_2)$ iff there exists $(A, h) \in d\text{Arg}(P)$ such that $A \subseteq A_1$ and $h$ and $h_2$ disagree. We say in this case that the argument $(A_1, h_1)$ is $d$-attacked by $(A_2, h_2)$ at $h$ or that $(A_2, h_2)$ $d$-attacks or $d$-rebuts $(A_1, h_1)$ at $h$. We also say that $(A_2, h_2)$ is a $d$-counter-argument of $(A_1, h_1)$.

Definition 29 $(A_1, L_1)$ is a proper $d$-defeater of $(A_2, L_2)$ (at literal $h$) if $(A_2, L_2) R_d^h (A_1, L_1)$ and there exists a sub-$d$-argument $(A, L)$ of $(A_2, L_2)$ such that $(A, L) \prec (A_1, L_1)$. We denote this by $(A_2, L_2) \leq_d (A_1, L_1)$.

$(A_1, L_1)$ is a blocking $d$-defeater of $(A_2, L_2)$ (at literal $h$) if $(A_2, L_2) R_d^h (A_1, L_1)$ and there exists a sub-$d$-argument $(A, L)$ of $(A_2, L_2)$ such that $(A, L) \not\prec (A_1, L_1)$ and $(A_1, L_1) \not\prec (A, L)$. Whenever this is the case we use the notation $(A_2, L_2) \equiv_d (A_1, L_1)$.

We call $d$-defeaters of an argument to all proper and blocking $d$-defeaters. If $(A_2, L_2)$ is a $d$-defeater of $(A_1, L_1)$ we write $(A_1, L_1) \leq_d (A_2, L_2)$.

We now proceed to analyze three cases where we can see the difference between the formalisms in action.

Spurious Attacks
Example 8 Using the program from example 6, we see that the equivalence class of arguments $B$ and $C$ is the same so the $d$-arguments are $(\{A\}, b), (\{B\}, d)$ and $(\{B\}, \sim b)$. These two last $d$-arguments are each other’s sub-$d$-arguments as well. This enables that $(\{A\}, b) d$-attack $(\{B\}, d$ or the literal $\sim b$, which does not appear in the deduction for $d$. Since $\sim b \notin C(B)$, this attack does not happen in the new framework.

Proper and blocking defeaters
Remark 7 Notice that our notion of sub-argument is more restrictive than that of (García and Simari 2004), since all that is required for $A$ to be a sub-$d$-argument of $B$ is $A \cap \Delta \subseteq B$, while we now demand that $A \subseteq B$. This eliminates some undesirable effects like the one in the following example.

Example 9 Let $P_2$ be the program with $\Pi_f = \{a\}$, $\Pi_r = \{b \rightarrow h, c \rightarrow h, b \wedge c \rightarrow h_1\}$, while $\Delta = \{a \sim b, a \sim c, a \sim \sim h_1\}$. Here we had that the $d$-argument $(\{a \sim \sim h_1\}, \sim h)$ $d$-attacks $(\{a \sim b, a \sim c\}, h_1)$ at literal $h$, but we have two sub-$d$-arguments at which the attack can happen: $(\{a \sim b\}, h)$ and $(\{a \sim c\}, h)$. We let $(\{a \sim b\}, h) \prec (\{a \sim \sim h_1\}, \sim h)$ when we obtain: $(\{a \sim b\}, h) \leq d (\{a \sim \sim h_1\}, \sim h)$ $(\{a \sim b\}, h) \leq d (\{a \sim \sim h_1\}, \sim h)$ $(\{a \sim b, a \sim c\}, h_1) \leq d (\{a \sim \sim h, \sim h\}$ $(\{a \sim c\}, h) \leq d (\{a \sim \sim h, \sim h\}$ $(\{a \sim c\}, h_1) \leq d (\{a \sim \sim h_1, \sim h\}$ $(\{a \sim b, a \sim c\}, h_1) \leq d (\{a \sim \sim h, \sim h\}$

So in this case $(\{a \sim \sim h\}, \sim h)$ was both a proper and a blocking $d$-defeater for the same argument and at the same literal.
Under the new definitions presented in this paper, the argument \( \{a, a \succ -b, a \succ -c, b \land c \rightarrow h_1\} \) for \( h_1 \) has no sub-arguments for \( h \), so the conflict does not occur.

**Discerning the facts**

When one uses d-arguments, the facts and strict rules used for a derivation are not recorded. This ends up meaning that all of them are implicit in each d-argument. One may as well consider a unique fact \( f_P \), logically equivalent to the conjunction of all the facts in a program \( P \), and obtain an equivalent program. With the proposed definition of arguments given here, the information on which facts are used to support a conclusion is recorded, and may later be used for comparing the relative weight of competing arguments. We think this adds to the flexibility of DELP, in particular when applied in the field of interacting agents (Capobianco, Chesnèvar, and Simari 2005), where different agents may trust information on facts in different degrees.

**Conclusions**

This paper introduced a new version of DELP in which the central notion of argument has been modified. It contains now both defeasible and facts and strict rules. Problems that arose from restricting arguments to include only defeasible rules disappear. On one hand, now each argument supports a unique literal, so the attack relation is more transparently defined. On the other, the derived relation of defeat (be it proper or blocking) obtains in an unambiguous way.

These slight modifications of the DELP formalism ensure a leaner non-monotonic inference mechanism, mainly because each argument now supports a unique literal. Furthermore, the information carried by each argument is richer than in previous versions.

The new system has not yet been implemented, so we make no claims about its computational cost. We suspect, however, that it might be less than the classical version of DELP because of the closer correspondence between arguments an literals. This should ease the calculation of the attack and defeat relations. Manipulating lists (as the annotated derivations) may also be easier than dealing with sets.

Finally, the explicitness of the representation promoted here facilitates the interaction of several sources of information. This feature is particularly appropriate in multi-agent environments.

**References**


