A Knowledge Representation Language for Defeasible Argumentation[1] [2]

Guillermo R. Simari   Alejandro J. García[3]

Grupo de Investigación en Inteligencia Artificial (GIIA)

Departamento de Ciencias de la Computación

Universidad Nacional del Sur

Av.Alem 1253 – (8000) Bahía Blanca, ARGENTINA

FAX: (54) (91) 553933   Phone: (54) (91) 20776 (ext. 208)

e-mail: `grs@criba.edu.ar`   `ccgarcia@criba.edu.ar`

Abstract

The goal of this work is to define a system for defeasible argumentation, as an extension of logic programming. Here, we define *Defeasible Logic Programs* (DLP), and a notion of defeasible inference from a DLP. The characteristics of a DLP allow to represent different flavors of the closed world assumption (CWA) just as program clauses, thus a CWA clause could be presented for some predicates and not for others. We will define an *argument* as a subset of a DLP, and therefore we can use some concepts of an argumentative system [12] in order to define the inference engine of the system. The semantics of a DLP is characterized by sets of positive, negative, undecided and unknown answers. Finally, we will define the behavior of an interpreter based upon this semantics.

[1]Published in proceedings of the XXI Latin American Conference of Informatic, Brasil, 1995.

[2]This work was partially supported by the Secretaría de Ciencia y Técnica, Universidad Nacional del Sur.

[3]Fellow of Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), República Argentina

A Knowledge Representation Language for Defeasible Argumentation

1. Introduction

Common sense reasoners tend to think in a defeasible way. Although logic programming has been widely used for declarative programming, it has limitations for defeasible reasoning. There have been advanced some extensions of logic programming that tried to capture some kind of defeasible reasoning (see [3, 5, 4]), but unfortunately, a considerable number of common situations cannot be represented in these extensions. On the other hand, defeasible argumentation has been developed with the aim of giving a formal framework to defeasible reasoning. Research on many aspects of argumentative systems has recently produced some interesting results (see [13, 12, 7, 14, 10]).

The goal of this work is to define a system for defeasible argumentation as an extension of logic programming. We will introduce defeasible logic programs (DLP), and a notion of defeasible inference from a DLP. The language is going to be defined in terms of program clauses, which will be separated in two disjoint subsets: one that represents strict (sound) knowledge, and other that contains defeasible (tentative) information.

In our language a *literal* "l" is an atom "a" or a negated atom "$\neg a$", as defined by Lloyd in [6]. In this work the symbol "$\neg$" will represent the classical negation (*i.e.*, $\neg\neg a = a$), and the symbol "*not*" will represent the negation as failure. The symbol "$\sim$" will be used to denote the complement of a literal with respect to classical negation (*i.e.*, $\sim l = \neg l$, and $\sim\neg l = l$).

Definition 1.1 : *Extended program clause* (EPC)
An extended program clause is a program clause as defined by Lloyd in [6] but it can contain both classical negation and negation as failure. It is a clause of the form $l \leftarrow p_1, \ldots, p_n$, where l is a literal, and each p_i ($n \geq 0$) is a literal or a literal preceded by the symbol *not* of negation as failure. If n=0 we denote $l \leftarrow$ TRUE, and we say that l is a *fact*. □

Definition 1.2 : *Defeasible program clause* (DPC)
A defeasible program clause is a clause of the form $l \prec p_1, \ldots, p_n$, where l is a literal, and each p_i ($n \geq 1$) is a literal or a literal preceded by the symbol *not* of negation as failure. We use the symbol "$\prec$" to distinguish a DPC from an EPC, because a DPC will be used to

represent defeasible knowledge, *i.e.*, tentative information that we can use if nothing is said to the contrary. A clause "$l \mathrel{-\!\!\prec} A$", must be read as: "reasons for believing in A are good reasons for believing in l". $\square$

All variables in program clauses are assumed to be universally quantified. We will use an EPC to represent sound (or not defeasible) information as "bird(X) $\leftarrow$ penguin(X)", and a DPC to represent defeasible knowledge as "fly(X) $\mathrel{-\!\!\prec}$ bird(X)". Program clauses must be interpreted as inference rules and not as conditionals, and they have not a truth value by themselves.

As a typographic convention, we will use lower case letters to denote literals (l), upper case ones for sets of literals (L), calligraphic upper case ones for sets of program clauses ($\mathcal{L}$), and the usual PROLOG conventions for program clauses. Sometimes we will write "$c \leftarrow A$" to represent a program clause where A is the antecedent conjunction .

Program clauses could contain two types of negation: on one hand, the classical negation "$\neg$" can be used, that will allow us to write clauses as "$\neg$carnivorous(X) $\leftarrow$ cow(X)" (cows are not carnivorous), or "$\neg$dangerous(X) $\mathrel{-\!\!\prec}$ $\neg$carnivorous (X)" to express that things that are not carnivorous normally are not dangerous (defeasible in many ways). On the other hand, we have the negation as failure represented by "*not*". This is very useful for representing clauses as "careful(X) $\mathrel{-\!\!\prec}$ *not* dangerous(X)" (be careful with things you do not know if they are dangerous), or "$\neg$cross-railway-tracks $\mathrel{-\!\!\prec}$ *not* $\neg$train-is-coming" (do not cross the railway tracks if you do not know if the train is coming). This sort of rules could not be written if we only use one type of negation.

Since program clauses could contain both classical negation and negation as failure, the closed world assumption (CWA) could be applied to a particular predicate in a particular clause. Therefore the CWA could be used for one predicate p just by writing "$\neg$p(X) $\leftarrow$ *not* p(X)" in the DLP. The advantage here is that we can use CWA for some predicates, and not for others. For example, it could be useful to have the rule "$\neg$dead(X) $\leftarrow$ *not* dead(X)" (*i.e.*, if you cannot prove that X is dead, assume that it is not dead). Furthermore, EPC allow others kinds of CWA: "p(X) $\leftarrow$ *not* $\neg$p(X)", "p(X) $\leftarrow$ *not* p(X)" , and "$\neg$p(X) $\leftarrow$ *not* $\neg$p(X)". On the other hand, if we use a defeasible clause instead of an EPC, another type of CWA could be represented: "$\neg$p(X) $\mathrel{-\!\!\prec}$ *not* p(X)", *i.e.*, the failure of the proof of $p(X)$ is a good reason to assume $\neg p(X)$. This subject is treated in more detail in section 5.

2. Defeasible Logic Programs

Definition 2.1 : *Defeasible logic program* (DLP)
A defeasible logic program is a finite set of EPC and DPC. Let $\mathcal{P}$ be a DLP, then we distinguish the subset $\mathcal{S}$ of EPC in $\mathcal{P}$, and the subset $\mathcal{D}$ of DPC in $\mathcal{P}$ for convenience. □

Definition 2.2 : *Consistency*
A set of program clauses $\mathcal{P}$ is consistent if we cannot defeasibly entail (see definition 2.4) a pair of complementary literals (with respect to classical negation). On the contrary, a set of program clauses $\mathcal{P}$ is inconsistent if a pair of complementary literals can be defeasibly entailed from $\mathcal{P}$. □

Given a DLP $\mathcal{P}$, the set $\mathcal{S}$ must be consistent, although the subset $\mathcal{D}$, and $\mathcal{P}$ itself (*i.e.*, $\mathcal{S} \cup \mathcal{D}$) may be inconsistent. Thus, a DLP can contain potentially-inconsistent information. In example 2.1, $\mathcal{S}$ is consistent, but $\mathcal{P}$ is not.

Example 2.1 :

bull(pepe) $\leftarrow$ TRUE
¬carnivorous(X) $\leftarrow$ bull(X)
has-horns(X) $\prec$ bull(X)
¬dangerous(X) $\prec$ ¬carnivorous(X)
dangerous(X) $\prec$ ¬carnivorous(X), has-horns(X)

Here we can defeasibly infer (see definition 2.4) bull(pepe), ¬ carnivorous(pepe), has-horns(pepe), ¬ dangerous(pepe), and dangerous(pepe). □

Definition 2.3 : *Goal*
A goal is a literal m. □

Definition 2.4 : *Defeasible proof for a goal m*
Given a DLP $\mathcal{P}$, a defeasible proof for a goal m from $\mathcal{P}$, is a finite set of ground EPC and DPC recursively defined as follows:

1. If there exists a fact "$c \leftarrow$ TRUE" in $\mathcal{P}$, such that m unifies with c (with mgu σ), then the set $\{c\sigma \leftarrow \text{TRUE}\}$ is a defeasible proof for m.
2. If there exists an EPC "$c \leftarrow L$" in $\mathcal{P}$, such that c unifies with m (with mgu σ), and there exists a defeasible proof $\mathcal{F}_i$ for each of the elements of $L\sigma$, then, $\{c\sigma \leftarrow L\sigma\} \cup (\bigcup_i \mathcal{F}_i)$ is a defeasible proof for m. In order to avoid cycles in the proof, the clause "$c \leftarrow L$" cannot appear in any of the sets $\mathcal{F}_i$.

3. If there exists a DPC "$c \multimap L$" in $\mathcal{P}$, such that c unifies with m (with mgu σ), and there exists a defeasible proof $\mathcal{F}_i$ for each of the elements of $L\sigma$, then, $\{c\sigma \multimap L\sigma\} \cup (\bigcup_i \mathcal{F}_i)$ is a defeasible proof for m. In order to avoid cycles in the proof, the clause "$c \multimap L$" cannot appear in any of the sets $\mathcal{F}_i$.
4. If m is a subgoal that has the negation as failure operator *not* (*i.e.*, $m = \textit{not}\ l$), and it does not exist a defeasible proof for the literal l, then the empty set is a defeasible proof for m.
5. If all the previous cases fail, then there does not exist a defeasible proof for m.

If there exists a defeasible proof for m from $\mathcal{P}$, then we say that $\mathcal{P}$ defeasibly entails m. □

Proposition 2.1 : *Given a* DLP $\mathcal{P}$*, the process of finding a defeasible proof for a goal* m *finishes in a finite amount of time.*
Proof: The number of clauses of $\mathcal{P}$ is finite, and each of them has a finite number of antecedents, and the process itself establishes conditions not to enter in infinite cycles, so it will be only a finite number of recursively calls in the previous definition. □

3. Knowledge Representation

In this section, we present some examples in order to show the language expressiveness for knowledge representation.

Example 3.1 :

bull(pepe) ← TRUE
carnivorous(tito) ← TRUE
dog(chicho) ← TRUE
¬carnivorous(X) ← bull(X)
¬dangerous(X) $\multimap$ ¬carnivorous(X)
dangerous(X) $\multimap$ *not* ¬dangerous(X)

In this example, as we know that "pepe" is not "carnivorous", we can defeasibly entail "¬dangerous(pepe)". But, as we do not know whether "chicho" or "tito" are dangerous or not, using the (prudence) rule "dangerous(X) $\multimap$ *not* ¬dangerous(X)" we can defeasibly entail "dangerous(tito)" and "dangerous(chicho)". □

Example 3.2 :

away(sailor1) $\leftarrow$ TRUE
away(sailor2) $\leftarrow$ TRUE
back(sailor2) $\leftarrow$ TRUE
$\neg$back(X) $\leftarrow$ *not* back(X)
lost(X) $-\!\!\!\prec$ away(X), $\neg$back(X)
look-for(X) $-\!\!\!\prec$ lost(X), *not* dead(X)
$\neg$look-for(X) $-\!\!\!\prec$ weather(very-bad)
weather(very-bad) $\leftarrow$ TRUE

From this DLP we can defeasibly entail "lost(sailor1)", but we cannot infer "lost(sailor2)" because we know he is back. □

The defeasible proof notion does not forbid inferring two complementary literals from a given DLP $\mathcal{P}$. In the example 3.2, we can defeasible entail "look-for(sailor1)" and "¬look-for(sailor1)". In order to allow only one of two complementary goals to be accepted as a new belief, we need an entailment criterium for choosing one of them.

Thus, we will define an *argument* as a DLP subset, and after that we will use the concepts of *justification* and *defeater* introduced in [12] in order to define an inference machine for this language. This will be developed in the next section.

4. DLP and Defeasible Argumentation

An Argumentative System [13, 8, 12, 14] is a formalization of the process of defeasible reasoning. An *argument* A for a hypothesis h is a tentative piece of reasoning that an agent would be inclined to accept as an explanation for h. An important aspect of argumentative systems concerns the question of how the hypothesis supported by an argument becomes accepted as part of the knowledge of the agent. We follow here the framework presented in [13] and [12] which accepts an argument A as a defeasible reason for a conclusion h if A is a *justification* for h.

The justification process involves the construction of an *acceptable argument* for h from the knowledge base of the system. To decide the acceptability of this argument, possible *defeaters* (see definition 7.3) are generated, which will be also tested for acceptability. Those defeaters that become *accepted* will be then compared with the original argument using a preference relation (*e.g.*, specificity), which defines a partial ordering among arguments. The original argument will be considered a *justification* (see definition 7.6) if it is "better" than every acceptable defeater. See appendix for further details.

In this work we will define an *argument* as a subset of a DLP, and after that we will use the concepts of *justification* and *defeater* in order to define the behavior of an interpreter for this language.

Definition 4.1 : *Argument*
Let $\mathcal{P}$ be a DLP, $\mathcal{S}$ the subset of EPC in $\mathcal{P}$, and $\mathcal{D}$ the subset of DPC in $\mathcal{P}$. An argument A for a literal h, is a subset of ground DPC of $\mathcal{D}$, such that:

1. There exists a defeasible proof for h from $\mathcal{S} \cup A$.
2. A is consistent with $\mathcal{S}$ (according to definition 2.2).
3. A is the minimal set (wrt set inclusion) that satisfies the two previous conditions.

If A is an argument for h, we also say that $\langle A, h\rangle$ is an argument structure. $\Box$

In example 2.1, the argument A={ dangerous(X) $\prec$ ¬carnivorous(X), has-horns(X); has-horns(X) $\prec$ bull(X) } is a justification for "dangerous(bull)". Although we have an argument B= {¬dangerous(X) $\prec$ ¬carnivorous(X) } for "¬dangerous(pepe)", B is not a justification because A defeats B. In example 3.2 we have arguments for both "look-for(sailor1)" and "¬look-for(salior1)", but neither of them have a justification (they are both blocking defeaters of each other).

Given a DLP, and using an argumentative system, we can identify a finite set of literals that have a justification. The elements of this set are the kind of believes a common sense reasoner answer positively in a consult.

Definition 4.2 : *Positive-answer set of a* DLP
Let $\mathcal{P}$ be a DLP. The positive-answer set of $\mathcal{P}$ is a finite set L of literals, such that for each $h \in L$, there exists an argument A that is a justification for h. $\Box$

Therefore, the positive-answer set of example 2.1 will be { bull(pepe), ¬carnivorous(pepe), has-horns(pepe), dangerous(pepe) }. On the other hand, in the same example, we cannot entail "¬dangerous(pepe)" because there exists a better argument for "dangerous(pepe)" ,*i.e.*, a proper defeater (see definition 7.3). In this case the answer for the query "¬dangerous(pepe)" might be negative. The negative-answer set of a DLP is characterized by the following definition.

Definition 4.3 : *Negative-answer set of a* DLP
Let $\mathcal{P}$ be a DLP. The negative-answer set of $\mathcal{P}$ is a finite set L of literals, such that for each $h \in L$, it holds one of the following conditions:

a) for each argument A for h, there exists at least a proper defeater of A.
b) it does not exist an argument for h, but there exists an argument that is a justification for $\sim h$.

□

Applying the condition (a) of the previous definition the literal "¬dangerous(pepe)" belongs to the negative-answer set of example 2.1. On the other hand, in example 4.1, "dangerous(ico)" belongs to the negative-answer set of this DLP because of the application of condition (b).

Example 4.1 :

$$\text{horse(ico)} \leftarrow \text{TRUE}$$
$$\neg\text{carnivorous(X)} \leftarrow \text{horse(X)}$$
$$\neg\text{dangerous(X)} \;—\!\!\prec\; \neg\text{carnivorous(X)}$$

□

There are also queries that can be neither positive nor negative. For example the query "look-for(sailor1)" in example 3.2. In that case the common sense reasoner is undecided, because, although it has an argument for "look-for(sailor1)", it also has an argument for "¬look-for(sailor1)", and it cannot be say if one argument is better than the other.

Definition 4.4 : *Undecided-answer set*
Let $\mathcal{P}$ be a DLP. The undecided-answer set of $\mathcal{P}$ is a finite set L of literals, such that for each $h \in L$, it holds one of the following conditions:

a) for each argument A of h, A has not proper defeaters, although it has a blocking defeater (*i.e.*, it has defeaters that interfere the justification, but we cannot prove they were better than A).
b) there is not argument for h but the following situation holds: (i) there exists a minimal set of literals J that cannot be entailed from $\mathcal{P}$, (ii) J and $\mathcal{P}$ allow the formation of an argument A for h, and (iii) for some $l \in J$, $\sim l \in \mathcal{S}$.

□

Finally, there exist queries for which we have not information at all, and their answers are simply unknown. For example the tomorrow lottery's winner number.

Definition 4.5 : *Unknown-answer set*
Let $\mathcal{P}$ be a DLP. The unknown-answer set of $\mathcal{P}$ is a possible infinite set of literals L, such that for each $h \in L$, h does not belong to any of the previous answer sets. □

Observation 4.1 : Given a DLP $\mathcal{P}$, its positive and negative-answer set are both consistent sets (*i.e.*, they not contain a pair of complementary literals). On the other hand, its undecided-answer set is normally an inconsistent one.

Once we have defined the four possible answer sets of a DLP, we will establish how a DLP interpreter must behave in order to resolve a given query.

Definition 4.6 : Given a DLP $\mathcal{P}$ and a goal m, a DLP interpreter might answer:

- YES, if m belongs to the positive-answer set of $\mathcal{P}$.
- NO, if m belongs to the negative-answer set of $\mathcal{P}$.
- UNDECIDED, if m belongs to the undecided-answer set of $\mathcal{P}$.
- UNKNOWN, if m belongs to the unknown-answer set of $\mathcal{P}$.

□

5. CWA clauses

As we have already said, program clauses could contain both classical negation ($\neg$) and negation as failure (*not*). In a DLP $\mathcal{P}$ the CWA of a predicate p could be expressed just by writing the clause "¬p(X) $\leftarrow$ *not* p(X)" in $\mathcal{P}$. Logic programming languages have the CWA for all predicates, *i.e.*, for any predicate q if the proof of q fail, they assume $\neg q$. The problem is that for some predicates in some circumstances the CWA could be a good police, but sometimes could not. The advantage here is that we can use CWA for some predicates, and not for others.

For example, it could be useful to have the rule "¬dead(X) $\leftarrow$ *not* dead(X)" (*i.e.*, if you cannot prove that X is dead, assume it is not dead), or the clause "¬back(X) $\leftarrow$ *not* back(X)" in example 3.2. On the other hand, if a pilot, ready to land a 747, uses the following rule: "¬busy-runway(X) $\leftarrow$ *not* busy-runway(X)" (if he does not know if the runway is busy, then assume it is not busy), the passengers could be in trouble if the pilot knows nothing about the runway. Thus, it seems to be convenient that the CWA could be used only for specific predicates, in a particular context.

The availability of using both types of negations allows us to define a new version of CWA: "p(X) $\leftarrow$ *not* $\neg$p(X)", *i.e.*, if you cannot prove that $\neg p$ is valid, then assume that p is valid. For example in the case of the pilot, the rule "busy-runway(X) $\leftarrow$ *not* $\neg$busy-runway(X)" would be better than the previous one. In example 3.1, the rule "dangerous(X) $\leftarrow$ *not* $\neg$dangerous(X)" was also used as a form of being prudent. This type of rule can also express optimism or hope, for example, "pass-exam(I) $\multimap$ *not* $\neg$pass-exam(I)".

In addition to the previous CWA clauses, where the antecedent is complementary of the consequent, there exist two CWA clauses that preserve the same literal: "p(X) $\leftarrow$ *not* p(X)" and "$\neg$p(X) $\leftarrow$ *not* $\neg$p(X)". Both of them have the same meaning.

On the other hand, if we use a defeasible clause instead of an EPC, another type of CWA could be represented: "$\neg$p(X) $\multimap$ *not* p(X)", *i.e.*, the failure of the proof of $p(X)$ is a good reason to assume $\neg p(X)$.

6. Conclusions

In this work, we have defined a defeasible logic programming language expressive enough to capture most of the examples we try to represent with this sort of reasoning. We have also shown that an argument could be seen as a subset of a DLP, thus, a DLP define an appropriate knowledge representation language for defeasible argumentation. In addition, the answer sets (based on defeasible argumentation concepts) represent a simple form of defining the behavior of a DLP interpreter.

Since classical negation and negation as failure are both available in the DLP language, the CWA could be represented as a program clause inside a DLP, and only for those predicates that would be convenient. Furthermore, new types of CWA could be used, improving the expressiveness of the language.

7. Appendix: a framework for defeasible argumentation

In this section the main definitions of a defeasible argumentation framework are introduced (see [13, 12] for further details). Here the set $\mathcal{K}$ corresponds to the set $\mathcal{S}$ of EPC.

Definition 7.1 : Two argument structures $\langle A_1, h_1\rangle$ and $\langle A_2, h_2\rangle$ *disagree*, denoted $\langle A_1, h_1\rangle \bowtie \langle A_2, h_2\rangle$, if and only if the set $\mathcal{K} \cup \{h_1, h_2\}$ is inconsistent.

Definition 7.2 : We say that $\langle A_1, h_1\rangle$ *counterargues* $\langle A_2, h_2\rangle$ at literal h, iff

1. There exists a subargument $\langle A, h\rangle$ of $\langle A_2, h_2\rangle$ such that $\langle A_1, h_1\rangle \bowtie \langle A, h\rangle$.
2. For every proper subargument $\langle S, j\rangle$ of $\langle A_1, h_1\rangle$, it is not the case that $\langle A_2, h_2\rangle$ counterargues $\langle S, j\rangle$.

Definition 7.3 : $\langle A_1, h_1\rangle$ *defeats* $\langle A_2, h_2\rangle$ at literal h, denoted $\langle A_1, h_1\rangle \gg_{\mathbf{def}} \langle A_2, h_2\rangle$, if and only if there exists a subargument $\langle A, h\rangle$ of $\langle A_2, h_2\rangle$ such that: $\langle A_1, h_1\rangle$ counterargues $\langle A_2, h_2\rangle$ at the literal h and

1. $\langle A_1, h_1\rangle$ is strictly more specific[4] than $\langle A, h\rangle$, or
2. $\langle A_1, h_1\rangle$ is unrelated by specificity to $\langle A, h\rangle$.

In case (1) $\langle A_1, h_1\rangle$ will be called a *proper* defeater, and in case (2) a *blocking* defeater. If $\langle A_1, h_1\rangle$ *defeats* $\langle A_2, h_2\rangle$, we will also say that $\langle A_1, h_1\rangle$ is a *defeater* for $\langle A_2, h_2\rangle$.

Definition 7.4 : Let $\langle A, h\rangle$ be an argument structure. A *dialectical tree* for $\langle A, h\rangle$, denoted $\mathcal{T}_{\langle A,\, h\rangle}$, is recursively defined as follows:

1. A single node containing an argument structure $\langle A, h\rangle$ with no defeaters (proper or blocking) is by itself a dialectical tree for $\langle A, h\rangle$. This node is also the root of the tree.

2. Suppose that $\langle A, h\rangle$ is an argument structure with defeaters (proper or blocking) $\langle A_1, h_1\rangle$, $\langle A_2, h_2\rangle$, ..., $\langle A_n, h_n\rangle$. We construct the dialectical tree for $\langle A, h\rangle$, $\mathcal{T}_{\langle A,\, h\rangle}$, by putting $\langle A, h\rangle$ in the root node of it and by making this node the parent node of the roots of the dialectical trees of $\langle A_1, h_1\rangle$, $\langle A_2, h_2\rangle$, ..., $\langle A_n, h_n\rangle$, *i.e.*, $\mathcal{T}_{\langle A_1,\, h_1\rangle}$, $\mathcal{T}_{\langle A_2,\, h_2\rangle}$, ..., $\mathcal{T}_{\langle A_n,\, h_n\rangle}$. If an unacceptable argument line gets formed (see [12]) during the construction of this tree, it suffices to clip the subtree rooted in the offending argument that violates a condition in the definition of acceptable argumentation line.

Definition 7.5 : Let $\mathcal{T}_{\langle A,\, h\rangle}$ be a dialectical tree for an argument structure $\langle A, h\rangle$. The nodes of $\mathcal{T}_{\langle A,\, h\rangle}$ can be recursively labeled as *undefeated nodes* (U-nodes) and *defeated nodes* (D-nodes) as follows:

1. Leaves of $\mathcal{T}_{\langle A,\, h\rangle}$ are *U-nodes.*
2. Let $\langle B, q\rangle$ be an inner node of $\mathcal{T}_{\langle A,\, h\rangle}$. Then $\langle B, q\rangle$ will be an *U-node* iff every child of $\langle B, q\rangle$ is a *D-node.* $\langle B, q\rangle$ will be a *D-node* iff it has at least an *U-node* as a child.

Definition 7.6 : Let $\langle A, h\rangle$ be an argument structure, and let $\mathcal{T}_{\langle A,\, h\rangle}$ be its associated dialectical tree. We will say that A is a *justification* for h (or simply $\langle A, h\rangle$ is a *justification*) iff the root node of $\mathcal{T}_{\langle A,\, h\rangle}$ is an U-node.

[4]We use here specificity as a comparison criterium, but any other partial order among argument structures can be used without changing the meaning of the system.

References

[1] García A. J., Chesñevar C. I., and Simari G. R. *Making Argument Systems Computationally Attractive.* Proc. XIII Int. Conf. of the Chilean Society for Computer Science, October 1993.

[2] García A. J., Chesñevar C. I., and Simari G. R. *Bases de Argumentos: su mantenimiento y revisión.* XIX Conferencia Latinoamericana de Informática. Buenos Aires, August 1993.

[3] Gelfond M. and Lifschitz V. *Logic Programs with Classical Negation.* Proc. of 7th. Int. Conf. on Logic Programming (ICLP) 1990

[4] Inoue K. *Extended Logic Programming with Default Assumptions.* Proc. of 8th. Int. Conf. on Logic Programming (ICLP) 1991.

[5] Kowalski R. and Sadri F. *Logic Programs with Exceptions.* Proc. of 7th. Int. Conf. on Logic Programming (ICLP) 1990

[6] Lloyd J. W. *Foundations of Logical Programming.* 2nd. edition, Springer-Verlag 1987

[7] Nute Donald, *Basic defeasible logic*, in *Intensional Logics for Programming*, Ed by Luis Fariñas del Cerro, Claredon Press – Oxford (c) 1992.

[8] J. L. Pollock. *Defeasible Reasoning. Cognitive Science*, 11:481–518, 1987.

[9] Poole D. *A logical framework for default reasoning.* Artificial Intelligence 36 (1988).

[10] Prakken H. *Logical Tools for Modelling Legal Arguments (Ph.D. Thesis).* Vrije University, Amsterdam, Holanda. January 1993.

[11] Simari G. R. , Chesñevar C. I., and García A. J. *Focusing Inference in Defeasible Argumentation.* IV Iberoamerican Congress on Artificial Intelligence, October 1994.

[12] Simari G. R. , Chesñevar C. I., and García A. J. *The Role of Dialectics in Defeasible Argumentation.* XIV Int. Conf. of Chilean Computer Science Society, November 1994.

[13] Simari G. R. and Loui R. P. *A Mathematical Treatment of Defeasible Reasoning and its Implementation.* Artificial Intelligence, 53: 125–157,1992.

[14] Vreeswijk G. *Studies in Defeasible Argumentation (Ph.D. Thesis).* Vrije University, Amsterdam, Holanda. March 1993.