An Argumentative Framework for Reasoning with Inconsistent and Incomplete Information

Alejandro J. García[1] and **Guillermo R. Simari**[1] and **Carlos I. Chesñevar**[1]

Abstract. We present here a knowledge representation language, where defeasible and non-defeasible rules can be expressed. The language has two different negations: *classical negation*, which is represented by the symbol "∼" used for representing contradictory knowledge; and *negation as failure*, represented by the symbol "`not`" used for representing incomplete information. Defeasible reasoning is done using a argumentation formalism. Thus, systems for acting in a dynamic domain, that properly handle contradictory and/or incomplete information can be developed with this language.

An *argument* is used as a defeasible reason for supporting conclusions. A conclusion q will be considered justified only when the argument that supports it becomes a *justification*. Building a justification involves the construction of a non-defeated argument $\mathcal{A}$ for q. In order to establish that $\mathcal{A}$ is a non-defeated argument, the system looks for *counterarguments* that could be *defeaters* for $\mathcal{A}$. Since defeaters are arguments, there may exist defeaters for the defeaters, and so on, thus requiring a complete dialectical analysis. The system also detects, avoids, circular argumentation. The language was implemented using an abstract machine defined and developed as an extension of the Warren Abstract Machine (WAM).

1 The language

Our language is defined in terms of two types of program clauses:

- *extended program clauses* [2](EPC): l `<-` $q_1, \ldots, q_n$.
- *defeasible program clauses*[3](DPC): l `-<` $q_1, \ldots, q_n$.

There are two different negations: *classical negation*, which is represented by the symbol "∼" used for representing contradictory knowledge; and *negation as failure*, represented by the symbol "`not`" used for representing incomplete information. In both kinds of clauses l is a literal (*i.e.*, a predicate "p" or a negated predicate "$\sim p$"), and each q_i ($n \geq 0$) is a literal, or a literal preceded by the symbol `not`. Thus, classical negation is allowed in the consequent of a clause, and negation as failure over literals is allowed in the antecedent. If $n = 0$, an EPC becomes "$l \leftarrow$ `true`." (or simply "l.") and is called a *fact*, whereas a DPC becomes "l —≺ `true`.", and is called a *presumption*.

We will use the usual PROLOG typographic conventions for an EPC, except that we will write "`head <- body`" rather than "`head :- body`"; and "`head -< body`" for a defeasible clause. An EPC is used to represent sound (*i.e.*, not defeasible) information such as: `bird(X) <- penguin(X)` which expresses that "all penguins are birds", whereas a DPC is used to represent defeasible knowledge such as: `fly(X) -< bird(X)` which expresses that "presumably, a bird can fly" or "usually, a bird can fly."

As mentioned earlier, program clauses can contain two types of negation. Classical negation (∼) can be used in clauses such as:

```
∼guilty(X) <- innocent(X).
∼free(X) -< ∼innocent(X).
```

to express that "an innocent is not guilty", and that "usually, if someone is not innocent, then it is not free." Negation as failure (`not`) may also be used in clauses such as:

```
innocent(X) -< not guilty(X).
∼cross-railway-tracks -< not ∼train-is-coming.
```

to express that "assume that someone is innocent whenever it has not been proven that s/he is guilty" and "generally, do not cross railroad tracks if it cannot be proven that no train is coming." This kind of rules could not be written with only one type of negation.

Operationally, the difference between the two negations is as follows: a query "∼`q`" succeeds when there exists a proof for "∼`q`." On the other hand a query "`not q`" will succeed when no proof can be found for "`q`". In fact, proving a negated literal "∼`p`" is carried out just as if the "∼" symbol were syntactically part of the predicate name, thereby treating "∼`p`" as an atomic predicate name. In this system, a proof is a formal derivation called an *argument*, which *justifies* a query q. Arguments and justifications will be introduced in the following section.

With two types of negations, the Closed World Assumption (CWA) of a predicate `p` could be expressed within the language, writing the clause

```
∼p(X) <- not p(X).
```

i.e., "`∼p(X)` will be derived whenever the proof of `p(X)` fails." Also, new forms of CWA can be written with the obvious

[1] Artificial Intelligence Research Group – Departamento de Ciencias de la Computación – Universidad Nacional del Sur – Av.Alem 1253 – (8000) Bahía Blanca, ARGENTINA – e-mail: { `ccgarcia, grs, ccchesne` }`@criba.edu.ar`

[2] We use this terminology following [8] because we allow classical negation as done there. However, the system presented here, can accommodate inconsistency whereas the language reported in [8] cannot.

[3] Given the similarity in use and syntax, we use the term 'clause' for this construction, even though it is not properly a clause, but a meta relation between the head and body of the rule.

ECAI 98 Workshop on Practical Reasoning and Rationality
13th European Conference on Artificial Intelligence
24th August 1998, Brighton, England.

interpretation:

```
p(X) <- not ∼p(X).
p(X) <- not p(X).
∼p(X) <- not ∼p(X).
```

Nevertheless, if a defeasible clause is used, a defeasible notion of CWA could be represented:

```
∼p(X) -< not p(X).
```

which expresses that "the failure of the proof of `p(X)` is a good reason to assume `∼p(X)`". Thus, the following clauses can be written:

```
∼dead(X) -< not dead(X).
dangerous(X) -< not ∼dangerous(X).
```

Definition 1.1 *A defeasible logic program (*DLP*) is a finite set of* EPC*s and* DPC*s.*

Let $\mathcal{P}$ be a DLP; then, we will distinguish the subset $\mathcal{S}$ of EPC in $\mathcal{P}$, and the subset $\mathcal{D}$ of DPC in $\mathcal{P}$.

Example 1.1 : Here follows a DLP that will be referred to in other examples:

```
fly(X) -< bird(X).
∼fly(X) -< chicken(X).
fly(X) -< chicken(X),scared(X).
chicken(coco) -< true.
penguin(petete) -< true.
∼dangerous(X) -< pet(X).
dangerous(X) -< tiger(X).
bird(X) <- chicken(X).
scared(coco).
bird(X) <- penguin(X).
∼fly(X) <- penguin(X).
pet(kitty).
tiger(kitty).
```

□

Given a DLP $\mathcal{P}$, a *defeasible derivation* for a query " `-< q`" is a finite set of EPC and DPC obtained by backward chaining from `q` as in a PROLOG program, using both strict and defeasible rules in the order specified by the DLP. The symbol "∼" is considered as part of the predicate when generating a defeasible derivation.

Example 1.2 : Using the DLP of example 1.1, there are defeasible derivations for each of the following queries:

```
-< ∼fly(petete)
-< fly(petete)
-< fly(coco)
-< ∼fly(coco)
```

□

It can be seen from Example 1.2, that the defeasible derivation notion does not forbid inferring two complementary literals from a given DLP $\mathcal{P}$. In order to allow only one of two complementary goals to be accepted as a sensible possibility, we need a criterion for choosing between the two.

In the next section the notion of *argument* will be introduced to allow the defeasible argumentation formalism developed in [20, 19] which will allow us to define an inference scheme for the language. But first, we need to explicate when a set of clauses is deemed consistent.

Definition 1.2 (Consistency) *A set of program clauses $\mathcal{A}$ is* consistent *if there is no defeasible derivation for any pair of complementary literals (with respect to classical negation "∼"). Conversely, a set of program clauses $\mathcal{A}$ is* inconsistent *if there are defeasible derivations for a pair of complementary literals.*

Given a DLP $\mathcal{P}$, the set $\mathcal{S}$ must be consistent, although the set $\mathcal{D}$, and hence $\mathcal{P}$ itself (*i.e.*, $\mathcal{S} \cup \mathcal{D}$), may be inconsistent. It is only in this form that a DLP may contain potentially inconsistent information.

Example 1.3 : Here follows a set of rules that supports inconsistent conclusions, because `fly(petete)` and `∼fly(petete)` can be derived.

```
penguin(petete) -< true.
fly(X) -< bird(X).
bird(X) <- penguin(X).
∼fly(X) <- penguin(X).
```

Observe also, that the DLP of example 1.1 is an inconsistent set of program clauses, although its associated set $\mathcal{S}$ is a consistent one. □

2 Arguments, Rebuttals and Defeaters

In this framework, answers to queries must be supported by *arguments*. However, arguments may be defeated by other arguments. A query `q` will succeed if the supporting argument for it is not defeated; it then becomes a *justification*. Before defining formally the notion of justification, we define arguments, counterarguments and defeaters.

Definition 2.1 (Argument) *An* argument $\mathcal{A}$ *for a query h, also denoted $\langle \mathcal{A}, h \rangle$, is a subset of* ground instances *of* DPC*s of $\mathcal{D}$, such that: (1) There exists a defeasible derivation for h from $\mathcal{S} \cup \mathcal{A}$, (2) $\mathcal{S} \cup \mathcal{A}$ is consistent, and (3) $\mathcal{A}$ is minimal with respect to set inclusion.*

The above definition is adapted from [20] to fit in this framework. It states that an argument is a consistent defeasible derivation for a given query h, using a minimal set of rules. Note that EPCs are not part of the argument.

Example 2.1 : Consider the DLP of example 1.1.
The query `-< ∼fly(coco)` has the argument:

$$\mathcal{A}_1 = \left\{ \begin{array}{l} \texttt{∼fly(coco) -< chicken(coco).} \\ \texttt{chicken(coco) -< true.} \end{array} \right\}$$

However, the query `-< fly(coco)` has two arguments:

$$\mathcal{A}_2 = \left\{ \begin{array}{l} \texttt{fly(coco) -< bird(coco).} \\ \texttt{chicken(coco) -< true.} \end{array} \right\}$$

$$\mathcal{A}_3 = \left\{ \begin{array}{l} \texttt{fly(coco) -< chicken(coco),scared(coco).} \\ \texttt{chicken(coco) -< true.} \end{array} \right\}$$

The query `-< ~fly(petete)` has the argument:

$$\mathcal{A}_4 = \{\ \texttt{penguin(petete) -< true.}\ \}$$

Finally, there is no argument for `-< fly(petete)` because its defeasible derivation is inconsistent with respect to $\mathcal{S}$ (see example 1.3). □

Definition 2.2 (Sub-argument) *An argument $\langle \mathcal{B}, q\rangle$ is a* sub-argument *of $\langle \mathcal{A}, h\rangle$ if $\mathcal{B} \subseteq \mathcal{A}$.*

Definition 2.3 (Counterargument or rebuttal) *We say that $\langle \mathcal{A}_1, h_1\rangle$* counterargues or rebuts *$\langle \mathcal{A}_2, h_2\rangle$ at literal h, if and only if there exists a sub-argument $\langle \mathcal{A}, h\rangle$ of $\langle \mathcal{A}_2, h_2\rangle$ such that the set $\mathcal{S} \cup \{h_1, h\}$ is inconsistent.*

An argument is in fact a *proof tree*, involving rules from $\mathcal{S}$ and $\mathcal{D}$. Hence, arguments will be depicted as triangles abstracting a tree shape [19]. The upper vertex of the triangle will be labeled with the argument's conclusion, and the argument will be associated with the triangle itself. Sub-arguments will be represented as smaller triangles inside a big one, which corresponds to the main argument at issue. Figure 1 shows the graphical representation of an argument $\langle \mathcal{A}_2, h_2\rangle$ with one sub-argument $\langle \mathcal{A}, h\rangle$, and a counterargument $\langle \mathcal{A}_1, h_1\rangle$.

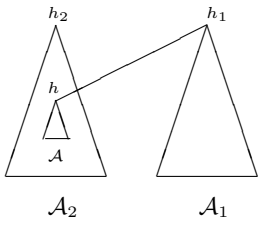

Figure 1. Argument $\langle \mathcal{A}_1, h_1\rangle$ counterargues $\langle \mathcal{A}_2, h_2\rangle$ at h

Example 2.2 : Continuing with Example 2.1, the argument $\mathcal{A}_1$ is a counterargument for both $\mathcal{A}_2$ and $\mathcal{A}_3$ (at `fly(coco)`), and also $\mathcal{A}_2$ and $\mathcal{A}_3$ are counterarguments for $\mathcal{A}_1$ (both at `~fly(coco)`). Note that the argument $\mathcal{A}_4$ has no counterarguments. □

As we mentioned before, the justification process for proving q involves the construction of a non-defeated argument $\mathcal{A}$ for q. In order to verify whether an argument is non-defeated, its associated counterarguments $\mathcal{B}_1$, $\mathcal{B}_2$, ... $\mathcal{B}_k$ are examined, each of them being a potential (defeasible) reason for rejecting $\mathcal{A}$. If any $\mathcal{B}_i$ is better than (or unrelated to) $\mathcal{A}$, then $\mathcal{B}_i$ is a candidate for defeating $\mathcal{A}$. If the argument $\mathcal{A}$ is better than $\mathcal{B}_i$ then $\mathcal{B}_i$ is not taken into account.

So we must clarify what makes an argument "better" than an other one. In this work, as a particular example, we will define a formal criterion called *specificity* which allows to discriminate between two conflicting arguments. However, the notion of *defeating argument* can be formulated independently of which particular argument-comparison criterion is used. Namely, if some $\mathcal{B}_i$ is better (*i.e.*, more specific, in our case) than $\mathcal{A}$, then $\mathcal{B}_i$ is called a *proper defeater* for $\mathcal{A}$; if neither argument is better than the other, a blocking situation occurs, and we will say that $\mathcal{B}_i$ is a *blocking defeater* for $\mathcal{A}$. This is an skeptical criterion that could be changed producing a different formal decision procedure.

Definition 2.4 (Defeating argument)
An argument $\langle \mathcal{A}_1, h_1\rangle$ defeats *an argument $\langle \mathcal{A}_2, h_2\rangle$ at literal h, if and only if there exists a sub-argument $\langle \mathcal{A}, h\rangle$ of $\langle \mathcal{A}_2, h_2\rangle$ such that $\langle \mathcal{A}_1, h_1\rangle$ counterargues $\langle \mathcal{A}_2, h_2\rangle$ at h, and either:*
(1) $\langle \mathcal{A}_1, h_1\rangle$ is strictly more specific than $\langle \mathcal{A}, h\rangle$ (then $\langle \mathcal{A}_1, h_1\rangle$ is a proper *defeater of $\langle \mathcal{A}_2, h_2\rangle$); or*
(2) $\langle \mathcal{A}_1, h_1\rangle$ is unrelated by specificity to $\langle \mathcal{A}, h\rangle$ (then $\langle \mathcal{A}_1, h_1\rangle$ is a blocking *defeater of $\langle \mathcal{A}_2, h_2\rangle$).*

The next definition characterizes specificity as defined in [14, 19] (adapted to fit in this framework). Intuitively, this notion of specificity favors two aspects in an argument: it favors an argument (1) with more information content and (2) with shorter derivations. In other words, an argument is deemed better that another if it is *more precise* or *more concise*. This notion is made formally precise in the next definition. We use the symbol "$\mid\!\sim$ " to denote a defeasible derivation (*i.e.*, $\mathcal{P} \mid\!\sim\ h$ means that h has a defeasible derivation from $\mathcal{P}$), and the symbol "$\vdash$" to denote a derivation where only EPCs are used. Let $\mathcal{S}_G$ be the maximal subset of $\mathcal{S}$ that does not contain facts. Let $\mathcal{F}$ be the set of literals in $\mathcal{P}$ that have a defeasible derivation.

Definition 2.5 (Specificity)
An argument $\mathcal{A}_1$ for h_1 is strictly more specific than *an argument $\mathcal{A}_2$ for h_2 (denoted $\langle \mathcal{A}_1, h_1\rangle \succ \langle \mathcal{A}_2, h_2\rangle$) if and only if:*
(1) For all $G \subseteq \mathcal{F}$: if $\mathcal{S}_G \cup G \cup \mathcal{A}_1 \mid\!\sim\ h_1$ and $\mathcal{S}_G \cup G \not\mid\!\sim\ h_1$, then $\mathcal{S}_G \cup G \cup \mathcal{A}_2 \mid\!\sim\ h_2$.
(2) There exists $G' \subseteq \mathcal{F}$ such that $\mathcal{S}_G \cup G' \cup \mathcal{A}_2 \mid\!\sim\ h_2$ and $\mathcal{S}_G \cup G' \not\mid\!\sim\ h_2$ and $\mathcal{S}_G \cup G' \cup \mathcal{A}_1 \not\mid\!\sim\ h_1$.

Example 2.3 : Continuing with Example 2.1, argument $\mathcal{A}_1$ is strictly more specific than the argument $\mathcal{A}_2$ because $\mathcal{A}_1$ does not use the EPC `bird(X) <- chicken(X)` and therefore is a more direct argument. However, argument $\mathcal{A}_3$ is strictly more specific than $\mathcal{A}_1$, because it contains more information. Then, $\mathcal{A}_1$ is a proper defeater for $\mathcal{A}_2$, and $\mathcal{A}_3$ is a proper defeater for $\mathcal{A}_1$. □

3 Dialectical Trees and Justifications

Since defeaters are arguments, there may exist defeaters for defeaters, and so on. This means that it is necessary to pursue argument supports to ascertain their well-foundedness. This justification process is called a *dialectical analysis*. It can be specified in the context of Logic Programming in the following way (here `\+` stands for PROLOG's negation as failure):

```
justify(Q) :- find_argument(Q,A), \+ defeated(A)
defeated(A) :- find_defeater(A,D), \+ defeated(D)
```

The above description leads in a natural way to the use of trees to organize our dialectical analysis. In order to accept an argument $\mathcal{A}$ as a justification for q, a tree structure can be generated. The root of the tree will correspond to the

argument $\mathcal{A}$ and every inner node will represent a defeater (proper or blocking) of its father. Leaves in this tree will correspond to non-defeated arguments. This structure is called a *dialectical tree.*

Definition 3.1 (Dialectical tree) *[19]*
Let $\mathcal{A}$ be an argument for h. A dialectical tree *for $\langle \mathcal{A}, h\rangle$, denoted $\mathcal{T}_{\langle \mathcal{A}, h\rangle}$, is recursively defined as follows:*
(1) A single node labeled with an argument $\langle \mathcal{A}, h\rangle$ with no defeaters (proper or blocking) is by itself a dialectical tree for $\langle \mathcal{A}, h\rangle$. This node is also the root of the tree.
(2) Suppose that $\langle \mathcal{A}, h\rangle$ is an argument with defeaters (proper or blocking) $\langle \mathcal{A}_1, h_1\rangle$, $\langle \mathcal{A}_2, h_2\rangle$, ..., $\langle \mathcal{A}_n, h_n\rangle$. We construct the dialectical tree for $\langle \mathcal{A}, h\rangle$, $\mathcal{T}_{\langle \mathcal{A}, h\rangle}$, by labeling the root node of with $\langle \mathcal{A}, h\rangle$ and by making this node the parent node of the roots of the dialectic trees for $\langle \mathcal{A}_1, h_1\rangle$, $\langle \mathcal{A}_2, h_2\rangle$, ..., $\langle \mathcal{A}_n, h_n\rangle$, i.e., $\mathcal{T}_{\langle \mathcal{A}_1, h_1\rangle}$, $\mathcal{T}_{\langle \mathcal{A}_2, h_2\rangle}$, ..., $\mathcal{T}_{\langle \mathcal{A}_n, h_n\rangle}$.

Nodes in the dialectical tree can be recursively marked as *defeated* or *undefeated* nodes (D-nodes and U-nodes respectively). Let $\mathcal{A}$ be an argument for a literal h, and $\mathcal{T}_{\langle \mathcal{A}, h\rangle}$ be its associated dialectical tree.

Definition 3.2 (Marking of a dialectical tree)
(1) Leaves of $\mathcal{T}_{\langle \mathcal{A}, h\rangle}$ are U-nodes.
(2) Let $\langle \mathcal{B}, q\rangle$ be an inner node of $\mathcal{T}_{\langle \mathcal{A}, h\rangle}$. Then $\langle \mathcal{B}, q\rangle$ will be an U-node *iff every child of $\langle \mathcal{B}, q\rangle$ is a* D-node*. The node $\langle \mathcal{B}, q\rangle$ will be a* D-node *iff it has at least an* U-node *as a child.*

This definition suggests a bottom-up marking procedure, through which we are able to determine if the root of a dialectical tree turns out to be marked as defeated or undefeated.

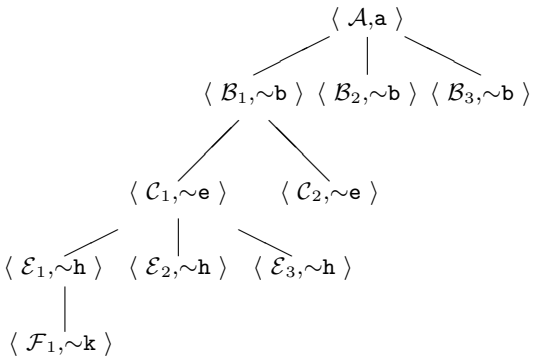

Figure 2. Dialectical tree for example 3.1

Example 3.1 : Consider the following DLP:

```
a -< b.      ~e -< l.     c ← true.
b -< c.      ~h -< k.     f ← true.
~b -< e.     ~b -< i.     i ← true.
e -< f.      k -< l.      l ← true.
~b -< c,f.   ~h -< c.     n ← true.
~e -< h.     ~h -< l.
h -< i.      ~k -< n,l.
```

Here the argument $\mathcal{A}$={ `a -< b. b -< c.` } for `a` can be built. Argument $\mathcal{A}$ has three defeaters attacking the literal `b`: $\mathcal{B}_1$= { `~b -< e. e -< f.` } $\mathcal{B}_2$= { `~b -< c,f.` } and $\mathcal{B}_3$= {`~b -< i.` }. $\mathcal{B}_2$ is a proper defeater for $\mathcal{A}$, the other two are blocking defeaters. Argument $\mathcal{B}_1$ has also two blocking defeaters attacking the literal `e`: $\mathcal{C}_1$= {`~e -< h. h -< i.`} and $\mathcal{C}_2$= { `~e -< l.`}. Argument $\mathcal{C}_1$ has three blocking defeaters: $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$, and finally $\mathcal{E}_1$has one proper defeater $\mathcal{F}_1$. The complete dialectical tree is shown in Figure 2. Figure 3 shows the same dialectical tree after applying the marking procedure of Definition 3.2. Note that nodes labeled with "#" need not to be considered in the analysis and pruning of the tree can be done. □

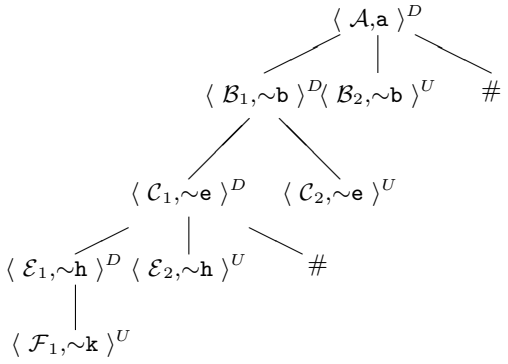

Figure 3. Marked dialectical tree for Figure 2 with pruning

Definition 3.3 (Justification) *Let $\mathcal{A}$ be an argument for a literal h, and let $\mathcal{T}_{\langle \mathcal{A}, h\rangle}$ be its associated dialectical tree. The argument $\mathcal{A}$ will be a justification for a literal h if the root of $\mathcal{T}_{\langle \mathcal{A}, h\rangle}$ is an U-node.*

If a query h has a justification, then it is considered 'justified', and the answer to the query will be YES. Nevertheless, there are several reasons for an argument not to be a justification: there may exist a non-defeated proper defeater, or a non-defeated blocking defeater, or there may be no argument at all. Therefore, in a DLP there are four possible answers for a query " $\prec$ `h`":

- YES, if there is a justification $\mathcal{A}$ for `h`.
- NO, if for each possible argument $\mathcal{A}$ for `h`, there exists at least one proper defeater for $\mathcal{A}$ marked as U-node.
- UNDECIDED, if for each possible argument $\mathcal{A}$ for `h`, there are no proper defeaters for $\mathcal{A}$ marked U-node, but there exists at least one blocking defeater for $\mathcal{A}$ marked U-node.
- UNKNOWN, if there exists no argument for `h`.

Finally, an example that shows all the concepts defined above is presented.

Example 3.2 : Given the DLP of Example 1.1, the following arguments can be built:

$\mathcal{A}_1$= { ∼fly(coco) -< chicken(coco). ; chicken(coco) -< true. }
for ∼fly(coco);

$\mathcal{A}_2$= { fly(coco) -< bird(coco). ; chicken(coco) -< true. }
for fly(coco);

$\mathcal{A}_3$= { fly(coco) -< chicken(coco),scared(coco). ; chicken(coco) -< true. }
for fly(coco);

$\mathcal{B}_1$= { ∼dangerous(kitty) -< pet(kitty). }
for ∼dangerous(kitty);

$\mathcal{B}_2$= { dangerous(kitty) -< tiger(kitty). }
for dangerous(kitty); and

$\mathcal{C}$={ penguin(petete) -< true. }
for ∼fly(petete)

If the query -< fly(coco) is submitted, the system first builds the argument $\mathcal{A}_2$ and looks for a defeater for it. Then, defeater $\mathcal{A}_1$ is found, so the system tries to find a defeater for $\mathcal{A}_1$, and then $\mathcal{A}_3$ is found. Since $\mathcal{A}_3$ has no defeaters, it becomes an U-node. Therefore $\mathcal{A}_1$ becomes a D-node and $\mathcal{A}_2$ becomes an U-node. Thus, $\mathcal{A}_2$ *is* a justification for fly(coco), and the answer for this query is YES.

The query "∼fly(coco)" is answered NO, because the argument $\mathcal{A}_1$ has the proper (non-defeated) defeater $\mathcal{A}_3$. The argument $\mathcal{B}_1$ has the blocking defeater $\mathcal{B}_2$, since $\mathcal{B}_2$ has no defeaters, then the query "dangerous(kitty)" is UNDECIDED. Note that the query "∼dangerous(kitty)" is also UNDECIDED. Finally the answer for "fly(petete)" is UNKNOWN because there is no argument for this query. □

4 Negation as Failure

As mentioned earlier, the language has two different negations: classical negation, which is represented by the symbol "∼" used for representing contradictory knowledge; and negation as failure, represented by the symbol "not" used for representing incomplete information. Operationally, the difference between the two negations is as follows: a query "∼q" succeeds when there exists a justification for "∼q." On the other hand a query "not q" will succeed when no justification for 'q" can be found. In our language, negation satisfies the coherent principle established in [2]: "If ∼p succeeds then not p also succeeds."

Example 4.1 : Consider the following DLP:

```
p(X) -< not q(X).
q(a).
q(X) -< r(X).
r(b).
r(c).
∼q(X) -< r(X),s(X).
s(c).
```

Here the query "∼q(c)" succeeds because it has a justification, however, there is no justification for "∼q(b)" (actually there is no argument for this query). The query "not q(d)" succeeds because there is no justification for q(d). However, since there is a justification for q(b), then the query "not q(b)" fails. Note that the query "not q(c)" succeeds, because although there is an argument for q(c), there is a defeater for it, so there is no justification for q(c). Thus, the queries p(c) and p(d) succeed, whereas p(a) and p(b) fail. □

5 Avoiding circular argumentation

A DLP is a finite set of program clauses, and therefore there is a finite number of arguments that may be involved in a dialectical tree. Nevertheless, we need to impose conditions in order to avoid cycles in this tree. In [19], it is shown that circular argumentation is a particular case of *fallacious argumentation*. There, a detailed analysis exposes different kinds of undesirable situations, and solutions are proposed accordingly. We next briefly present the problems and their solutions and refer to [19] for details.

In order to analyze fallacious argumentation, it is useful to see a dialectical tree as a set of *argumentation lines*. Following [19], this notion is formally defined as follows. Let $\langle \mathcal{A}_0, h_0 \rangle$ be an argument, and let $\mathcal{T}_{\langle \mathcal{A}_0, h_0 \rangle}$ be its associated dialectical tree.

Definition 5.1 (Argumentation line) *Every path λ from the root $\langle \mathcal{A}_0, h_0 \rangle$ to a leaf $\langle \mathcal{A}_n, h_n \rangle$ in $\mathcal{T}_{\langle \mathcal{A}_0, h_0 \rangle}$, denoted λ = $[\langle \mathcal{A}_0, h_0 \rangle, \langle \mathcal{A}_1, h_1 \rangle, \langle \mathcal{A}_2, h_2 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$, is called an* argumentation line *for h_0.*

In each argumentation line $\lambda = [\langle \mathcal{A}_0, h_0 \rangle, \langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_i, h_i \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$, the argument $\langle \mathcal{A}_0, h_0 \rangle$ is supporting the main query h_0, and every argument $\langle \mathcal{A}_i, h_i \rangle$ defeats its predecessor $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$. Therefore, for $k \geq 0$, $\langle \mathcal{A}_{2k}, h_{2k} \rangle$ is a *supporting* argument for h_0 and $\langle \mathcal{A}_{2k+1}, h_{2k+1} \rangle$ is an *interfering* argument for h_0. In other words, every argument in the line either supports h_0's justification or interferes with it. Therefore, an argumentation line can be split in two disjoint sets: λ_S of supporting arguments, and λ_I of interfering arguments. Thus, an argumentation line λ can be construed as an alternate sequence of supporting and interfering arguments as in any ordered debate. In a dialectical tree, there are as many argumentation lines as leaves in the tree, and each can end up in a supporting or an interfering argument.

We next present three types of fallacious argumentation, and establish necessary and sufficient conditions to avert them.

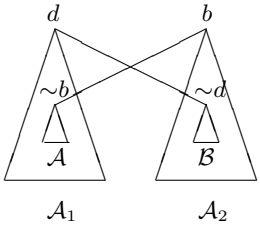

Figure 4. Reciprocal defeaters

The first problematic situation is shown in Figure 4. This happens when a pair of arguments defeat each other. In this

case, $\langle\ \mathcal{A}_1,d\ \rangle$ defeats $\langle\ \mathcal{A}_2,b\ \rangle$, attacking the subargument $\langle\ \mathcal{B},\sim d\ \rangle$, but $\langle\ \mathcal{A}_2,b\ \rangle$ also defeats $\langle\ \mathcal{A}_1,d\ \rangle$ attacking the subargument $\langle\ \mathcal{A},\sim b\ \rangle$. Clearly, this situation is nonsensical as it leads to an infinite argumentation line. The first condition expressed in Definition 5.3 prevents this situation.

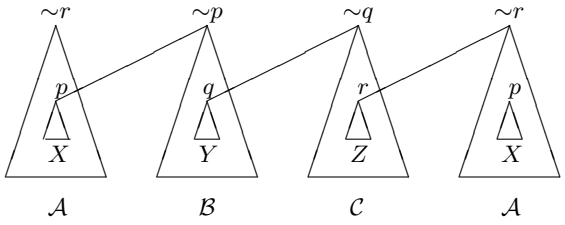

Figure 5. Contradictory argumentation line

Figure 5 shows a case where the same argument ($\mathcal{A}$) becomes both a supporting and an interfering argument of itself. This too is a nonsensical situation as it arises because the supporting argument $\mathcal{C}$ has a subargument Z for the literal r, which is contradictory with the purpose of arguing in favor of $\sim r$ (argument $\mathcal{A}$). An argument like $\mathcal{C}$ ought to be avoided in a sound argumentation line. Clearly, there should be agreement among supporting arguments (respectively interfering) in any argumentation line. This underlies the second condition in Definition 5.3. This is expressed formally based on the notion of argument *concordance* as proposed in [19] and recalled next. Supporting (respectively interfering) arguments should be mutually *concordant* so as to make the dialectical process of argumentation coherent.

Definition 5.2 (Concordance)
Two arguments $\langle \mathcal{A}_1, h_1\rangle$ *and* $\langle \mathcal{A}_2, h_2\rangle$ *are* concordant *iff the set* $\mathcal{S} \cup \mathcal{A}_1 \cup \mathcal{A}_2$ *is consistent. More generally, a set of arguments* $\{\langle \mathcal{A}_i, h_i\rangle\}_{i=1}^{n}$ *is concordant iff* $\mathcal{S} \cup \bigcup_{i=1}^{n} \mathcal{A}_i$ *is consistent.*

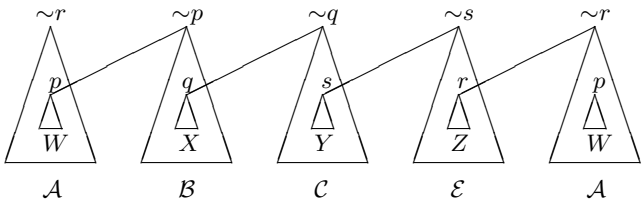

Figure 6. Circular argumentation

Finally, Figure 6 shows an example of circular argumentation, where the same argument $\mathcal{A}$ is introduced again in the line as a supporting argument. This is avoided by the third condition in Definition 5.3, which disallows the more general problematic situation where a subargument of an earlier argument is reintroduced further down the argumentation line.

These three situations are averted by requiring that all argumentation be addressed be *acceptable* as defined below:

Definition 5.3 (Acceptable argumentation line) *[19] Let* $\lambda = [\langle \mathcal{A}_0, h_0\rangle, \langle \mathcal{A}_1, h_1\rangle, \ldots, \langle \mathcal{A}_i, h_i\rangle, \ldots, \langle \mathcal{A}_n, h_n\rangle]$ *be an argumentation line,* λ *is an* acceptable argumentation line *iff:*
(1) For every defeater $\langle \mathcal{A}_i, h_i\rangle$ *and every proper subargument* $\langle \mathcal{B}, q\rangle$ *of* $\langle \mathcal{A}_i, h_i\rangle$, $\langle \mathcal{B}, q\rangle$ *and* $\langle \mathcal{A}_{i-1}, h_{i-1}\rangle$ *are concordant; that is,* $\mathcal{S} \cup \{q, h_{i-1}\}$ *must be consistent.*
(2) The sets λ_S *of supporting arguments, and* λ_I *of interfering arguments of* λ *must each be concordant sets of arguments.*
(3) No argument $\langle \mathcal{A}_k, h_k\rangle$ *in* λ *is a subargument of an earlier argument* $\langle \mathcal{A}_i, h_i\rangle$ *of* λ *(*$i < k$*).*

Hence, with these conditions averting undesirable situations, an *acceptable dialectical tree* is a dialectical tree where every argumentation line is acceptable. Then the notion of justification can be properly defined as follows [19].

Definition 5.4 (Justification) *The argument* $\langle \mathcal{A}, h\rangle$ *is a justification for* h *iff its associated dialectical tree is acceptable and the root node of* $\mathcal{T}_{\langle \mathcal{A},\, h\rangle}$ *is a* U*-node.*

6 Implementation

In order to develop an efficient defeasible argumentation system, an abstract machine called JAM (Justification Abstract Machine) [7] has been designed as an extension of the Warren's abstract machine (WAM) [1]. The JAM architecture has an instruction set, a memory structure and a set of registers for building arguments, counterarguments, and in this form obtaining justifications for queries. A compiler for defeasible logic programs was developed. It takes a defeasible logic program as its input and produces a sequence of JAM instructions as its output. The JAM was built as a virtual machine, and an interpreter for defeasible logic programs was developed over this machine.

7 Related Work

Other formalisms for defeasible argumentation have been separately developed. In [6] P. Dung has proposed a very abstract and general argument-based framework, where he completely abstracts from the notions of argument and defeat. In contrast we have defined an object language for representing knowledge and a concrete notion of argument and defeat, Dung's approach assumes the existence of a set of arguments ordered by a binary relation of defeat. However, he defines various notions of 'argument extensions', which are intended to capture various types of defeasible consequence.

Inspired by legal reasoning, H. Prakken and G. Sartor [17, 18] have developed an argumentation system that like us, uses the language of extended logic programming. They introduce a *dialectical proof theory* for an argumentation framework fitting the abstract format developed by Dung, Kowalski *et al.* [6, 4]. However, since they are inspired by legal reasoning, the protocol for dispute is rather different from our dialectical approach. A proof of a formula takes the form of a *dialogue tree*, where each branch of the tree is a dialogue between a *proponent* and an *opponent*. Proponent and opponent have different rules for introducing arguments, leading to an asymmetric dialogue. Later, Prakken [16] generalized the system to default logic's language.

R. Kowalski and F. Toni [9] have outlined a formal theory of argumentation, in which defeasibility is stated in terms of

non-provability claims. They argue that defeasible reasoning with rules of the form P if Q can be understood as "exact" reasoning with rules of the form P if Q and S cannot be shown, where S stands for one or more defeasible "non-provability claims".

Other related works are those by Vreesvijk [21], Bondarenko [3], Pollock [13], Loui [10], and Nute [11, 12]. The interested reader is referred to the following surveys in defeasible argumentation: Prakken & Vreesvijk [15], and Chesñevar *et al.* [5].

8 Conclusions

We have presented a knowledge representation language, that uses an argumentation formalism for defeasible reasoning, Defeasible rules allow to represent tentative knowledge, but also strong rules can be used. Since classical negation and negation as failure are both available in the language, contradictory and incomplete information can be represented. Several forms of CWA can be represented directly within the language.

Conclusions are supported by arguments, but when contradictory information is reached during a derivation, the defeasible argumentation formalism provides a way for deciding between competing arguments. If new information arises, then new arguments could be constructed and previous conclusions could be withdrawn. Thus, a correct behavior for a dynamic domain is obtained.

In order to develop efficient defeasible systems, the language was implemented using an abstract machine defined and implemented as an extension of the Warren Abstract Machine (WAM).

Acknowledgments

Alejandro J. García wishes to thank especially Hassan Aït-Kaci for many helpful discussions and suggestions. The authors are also grateful to Henry Prakken for helpful comments, and the anonymous referees for their suggestions. This work was partially supported by CONICET and Secretaría de Ciencia y Técnica UNS.

REFERENCES

[1] H. Aït-Kaci, *Warren's Abstract Machine—A Tutorial Reconstruction*, MIT Press, 1991.

[2] José Alferes and Luis M. Pereira, 'Contradiction: when avoidance equals removal (part i and ii)', in *Proc. of Ext. of Logic Programming, 4th International Workshop ELP'93 St. Andrews U.K*, (March 1993).

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

[4] A. Bondarenko, F. Toni, and R.A. Kowalski, 'An assumption-based framework for non-monotonic reasoning', *Proc. 2nd. International Workshop on Logic Programming and Non-monotonic Reasoning*, 171–189, (1993).

[5] C. I. Chesñevar, A. Maguitman, and R.P.Loui, 'Logical models of arguments', *submitted to ACM Computing Surveys*, (1998).

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

[7] Alejandro J. García, *Defeasible Logic Programming: Definition and Implementation* (MSc Thesis), Departamento de Cs. de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina, July 1997.

[8] M. Gelfond and V. Lifschitz, 'Logic programs with classical negation', in *Proceedings of the 7th International Conference on Logic Programming*, eds., D. Warren and P. Szeredi, pp. 579–597. MIT Press, (1990).

[9] Robert A. Kowalski and Francesca Toni, 'Abstract argumentation', *Artificial Intelligence and Law*, **4**(3-4), 275–296, (1996).

[10] Ronald P. Loui, Jeff Norman, Joe Altepeter, Dan Pinkard, Dan Craven, Jessica Lindsay, and Mark Foltz, 'Progress on room 5: A testbed for public interactive semi-formal legal argumentation', in *Proc. of the 6th. International Conference on Artifcial Intelligence and Law*, (July 1997).

[11] D. Nute, 'Basic defeasible logic', in *Intensional Logics for Programming*, ed., Luis Fariñas del Cerro, Clarendon Press, Oxford, (1992).

[12] D. Nute, 'Defeasible logic', in *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol 3, Nonmonotonic Reasoning and Uncertain Reasoning*, eds., C.J. Hogger D.M. Gabbay and J.A.Robinson, 355–395, Oxford University Press, (1994).

[13] John L. Pollock, *Cognitive Carpentry: A Blueprint for How to Build a Person*, Massachusetts Institute of Technology, 1995.

[14] David L. Poole, 'On the Comparison of Theories: Preferring the Most Specific Explanation', in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 144–147. IJCAI, (1985).

[15] H. Prakken and G. Vreesvijk, 'Logical systems for defeasible argumentation (to appear)', in *Handbook of Philosophical Logic, second edition*, ed., Gabbay, (1998).

[16] Henry Prakken, *Logical Tools for Modelling Legal Argument. A Study of Defeasible Reasoning in Law*, Kluwer Law and Philosophy Library, 1997.

[17] Henry Prakken and Giovanni Sartor, 'A system for defeasible argumentation, with defeasible priorities', in *Proc. of the International Conference on Formal Aspects of Practical Reasoning, Bonn, Germany*. Springer Verlag, (1996).

[18] Henry Prakken and Giovanni Sartor, 'Argument-based logic programming with defeasible priorities', *Journal of Applied Non-classical Logics*, **7**(25-75), (1997).

[19] Guillermo R. Simari, Carlos I. Chesñevar, and Alejandro J. García, 'The role of dialectics in defeasible argumentation', in *Anales de la XIV Conferencia Internacional de la Sociedad Chilena para Ciencias de la Computación*. Universidad de Concepción, Concepción (Chile), (November 1994).

[20] Guillermo R. Simari and Ronald P. Loui, 'A Mathematical Treatment of Defeasible Reasoning and its Implementation', *Artificial Intelligence*, **53**, 125–157, (1992).

[21] Gerard A.W. Vreeswijk, 'Abstract argumentation systems', *Artificial Intelligence*, **90**, 225–279, (1997).