Modeling Defeasibility in an Extended Logic Programming Setting Using an Abstract Argumentation Framework

Carlos Iván Chesñevar and Guillermo Ricardo Simari

Departamento de Ciencias de la Computación
Universidad Nacional del Sur
Av. Alem 1253 – (8000) Bahía Blanca – REPÚBLICA ARGENTINA
Tel./Fax: (+54) (291) 459 5135 / 5136
Email: {ccchesne, grs}@criba.edu.ar

KEY WORDS: defeasible argumentation, extended logic programming

Abstract. Defeasible argumentation has become a well-known approach to model commonsense reasoning. This has motivated the development of several alternative argumentative frameworks. The MTDR argumentative framework [SL92, SCG94] has proven to be a powerful approach to formalize defeasible argumentation, particularly in a logic-programming setting called defeasible logic programming (DLP) [Gar97]. Independently, the abstract argumentation framework developed by Bondarenko et al. [BDKT97] provides a sound theoretical basis for considering different kinds of non-monotonic reasoning systems. In recent work [KT99], the main focus has been on capturing semantics of logic programming in argumentative terms. This paper discusses a uniform way of conceptualizing defeasible rules and strict rules (used in defeasible logic programming) using extended logic programming under an abstract argumentation framework. As a result, we get an alternative formalization of DLP which allows a natural comparison of the well-founded semantics of normal programs (defined in argumentative terms) and the procedural semantics associated with defeasible logic programs.

1 Introduction and Motivations

In recent works it has been shown that Logic Programming (LP) with negation as failure (NAF) as well as many other formalisms (Reiter’s default logic [Rei80], Moore’s autoepistemic logic [Moo85], etc.) can be understood in an uniform way within an abstract argumentation framework [BDKT97]. In general, such an argumentation framework consists of a theory P in some background monotonic logic, which is extended by a set of candidate hypotheses $\mathcal{H}$, and a (binary) relation attacks between sets of hypothesis, satisfying certain fundamental properties. Different instances of this abstract argumentation framework can be shown to correspond to different non-monotonic formalisms.
In *A Mathematical Treatment of Defeasible Reasoning* [SL92] (MTDR for short), a framework for defeasible argumentation was introduced, which became later enriched through a number of dialectical considerations [SCG94], evolving into a refined, *dialectics*-based argumentation system [SCG94, ?]. This resulting framework has come to be a useful approach for further research because of its simplicity and expressive power. In particular, *defeasible logic programming* [Gar97] (DLP for short) was developed using MTDR as a theoretical basis.

In this paper we will recast DLP under the abstract argumentation framework. We consider how DLP arguments can be rewritten in terms of set of hypotheses by conceptualizing defeasible rules as ‘exact’ rules, as suggested by Kowalski & Toni in [?]. This results in an extended logic programming setting for conceptualizing defeasible logic programs. In this new setting we are able to subsume the notions of “negation as failure to justify” and defeasibility of rules by using a single non-provability operator “not”. We show some interesting relations between the well-founded semantics in normal programs (which can be modelled in argumentative terms under the abstract argumentation framework) and the semantics of defeasible logic programs recast in our extended logic programming setting.

The paper is structured as follows. First, in section 2, we present a brief summary of the main definitions of the abstract argumentation framework [BDKT97] needed to formalize well-founded semantics for normal programs in argumentative terms [KT99]. Then, in section 3, we will summarize the main definitions of MTDR in terms of defeasible logic programming (DLP). In section 4 we will show how to recast defeasible program clauses into an extended logic programming format, which makes us easier to compare semantics issues of DLP with respect to normal programs. The compared features will be analyzed and discussed in section 5. Finally, in section 6, we will present the main conclusions that have been obtained.

2 Abstract Argumentation Framework: Fundamentals

An *argumentative framework* [BDKT97, KT99] consists of a *theory* $P$ expressed in a background monotonic logic, a *set of hypotheses* $H$, and a binary *attack relationship* between sets of hypotheses $A, A' \subseteq H$, such that the following properties hold:

- No set of hypotheses $A$ attacks the empty set.
- The *attack* relationship is monotonic, that is, for every $A, A', B, B' \subseteq H$, if $A$ attacks $B$, it holds that
  - if $A \subseteq A'$, then $A'$ attacks $B$
  - if $B \subseteq B'$, then $A$ attacks $B'$
- The *attack* relationship is compact, *i.e.*, for every $A, B \subseteq H$, if $A$ attacks $B$, then there exists a finite $A', A' \subseteq A$, such that $A'$ attacks $B$.

Different instances of this abstract argumentation framework (AAF) can be shown to correspond to different non-monotonic formalisms. Logic programming
is a particularly interesting instance of the AAF, being considered in detail in [KT99]. If $P$ is a normal logic program, then the set of hypothesis associated with $P$ can be defined as

$$\mathcal{H}(P) = \{ \text{not } p \mid p \text{ is a variable-free atom in the Herbrand base of } P \}$$

and the background monotonic logic is the classical logic of (definite) Horn programs. These programs are obtained by replacing each negative literal $\text{not } p$ in the program $P$ by a new positive atom $\text{not } p$. Following [KT99], we will denote the resulting program using the same symbol $P$, and the consequence relation of this logic by $\models$.

Given a program $P$, a conjunction of (variable-free) literals $G$ is a non-monotonic consequence of $P$ wrt some semantics iff there exists a set of hypothesis $\Delta \subseteq \mathcal{H}(P)$, allowed by the semantics, such that $P \cup \Delta \models G$. Different semantics for LP allow different sets of hypothesis. Most of these semantics can be formulated by a single attacking relation between sets of (variable-free) negative literals.\(^1\)

**Definition 1.** (Attack). A set of hypothesis $A$ attacks another set $B$ (on an hypothesis $\text{not } p$) iff $P \cup A \models p$ for some $\text{not } p$ in $B$.

Various argumentation semantics [BDKT97] (such as admissibility, stable and well-founded models semantics) can be defined by imposing different restrictions on the acceptance of a given set of hypothesis. Thus, well-founded acceptability can be (informally) specified in this setting as follows:

**Definition 2.** (Well-founded acceptability). A set of hypothesis $A$ is well-founded acceptable (wf-acceptable) iff for all sets of hypothesis $B$, if $B$ attacks $A$, then there exists a set $C$ such that $C$ attacks $B$, and $C$ is wf-acceptable.

The base case in this recursive definition is the following: a set $A$ is wf-acceptable iff no set of hypotheses $B$ attacks $A$.

**Example 1.** Consider the logic program:\(^2\)

$$p \leftarrow \text{not } q$$

$$q \leftarrow \text{not } t$$

Then $\Delta_0 = \emptyset$ and $\Delta_1 = \{ \text{not } t \}$ are wf-acceptable, since no set of hypotheses attacks them.\(^3\) Therefore, the set $\Delta_1 = \{ \text{not } p \}$ is wf-acceptable, since it can be defended by the wf-acceptable set $\Delta_0$. It also holds that $\Delta_0 \cup \Delta_1$ is wf-acceptable.

Kakas & Toni [KT99] show how to formulate different argumentative semantics in terms of so-called pre-trees. Nodes in pre-trees are set of hypotheses, where each node is labelled either as attack or defense. Their approach allows

---

1. Note that the attack relationship does not state a preference criterion between sets of hypothesis.
2. This example was taken from [KT99].
3. As a special case, the empty set is always wf-acceptable.
describing different semantics for logic programming by restricting nodes which can appear in the branches of a pre-tree. In our analysis we will concentrate on the well-founded semantics (wf-semantics) of normal logic programs, which can be described in terms of well-founded pre-trees, defined as follows:

**Definition 3.** (Well-founded pre-trees). A well-founded pre-tree for a set of hypotheses \( \Delta \subseteq H \) is a labelled tree \( T \), where

1) Each node is a set of hypotheses, labelled as attack or defence, but not both.

2) The root \( \Delta \) is labelled as defence.

3) Each node \( A \) labelled as attack has exactly one child \( D \) labelled as defence, such that \( D \) attacks \( A \).

4) Each node \( D \) labelled as defence has as children nodes all those sets of hypotheses \( A \) such that \( A \) attacks \( D \).

5) Each branch in \( T \) is finite.

The next theorem establishes an important equivalence between the notion a set of hypotheses \( \Delta \) being well-founded and building a pre-tree with root \( \Delta \).

**Theorem 1.** [KT99] Let \( P \) be a normal logic program, and \( \Delta \subseteq H(P) \). A set of hypothesis \( \Delta \) is wf-acceptable iff there exists a well-founded pre-tree with root \( \Delta \).

3 MTDR and Defeasible Logic Programming

Next we will present the basic definitions of defeasible argumentation according to the MTDR formalization. We will use defeasible logic programming [Gar97] as a basis.

**Definition 4.** (Defeasible logic program). A defeasible logic program (DLP) is a 2-uple \( (S, D) \), where \( S \) is a finite set of defeasible rules, and \( D \) is a finite set of strong rules.

A strong rule has the form \( L_0 \leftarrow L_1, \ldots, L_n, \quad n \geq 0 \) (if \( n = 0 \) then \( L_0 \leftarrow \) is a fact) where \( L_0, \ldots, L_n \) are literals in an extended logic programming sense (that is, a literal is a positive atom \( p \) or a negated atom \( \neg p \), where \( \neg \) denotes strong negation). Literals \( L_1, \ldots, L_n \) are possibly preceded by the symbol “not” denoting default negation.

A defeasible rule has the form \( L_0 \leftarrow L_1, \ldots, L_n, \) and the same conditions as above apply (if \( n = 0 \) then \( L_0 \leftarrow \) is a presumption).

Syntactically, the symbol “\( \neg \) ” is all that distinguishes a defeasible rule from a strong rule. Defeasible rules account for tentative information that can be used if nothing could be posed against it. A query in DLP is understood analogously as a query in logic programming, except that both strong and defeasible rules can be used in the derivation.

A DLP \( P = (S, D) \) is said to be contradictory iff it is possible to defeasibly derive a pair of complementary literals \( p \) and \( \neg p \). It is assumed that the set \( S \) of strong rules is non-contradictory, although \( P \) itself may be contradictory.
Definition 5. (Argument in DLP). Given a DLP $P$, an argument $A$ for a query $q$, denoted $(A, q)$, is a subset of ground instances of defeasible rules of $P$, such that: 1) there exists a defeasible derivation for $q$ from $S \cup A$; 2) $S \cup A$ is non-contradictory; 3) $A$ is minimal wrt to set inclusion (i.e., there is no $A' \subset A$ such that $A'$ satisfies 1).

Arguments can be in conflict in different ways. This is formally captured by the notion of counterargument and defeat, as follows:

Definition 6. (Counterargumentation. Defeat). Given two arguments $(A_1, h_1)$, and $(A_2, h_2)$, we say that
- $(A_1, h_1)$ counterargues $(A_2, h_2)$ at literal $h$ iff there exists a sub-argument $(A_i, h)$ of $(A_e, h_2)$ such that the set $S \cup \{h_1, h\}$ is contradictory.
- $(A_1, h_1)$ defeats $(A_k, h_2)$ at literal $h$ iff $(A_1, h_1)$ counterargues $(A_k, h_2)$ at literal $h$, and $(A_1, h_1)$ strictly more specific than (or unrelated by specificity to) $(A_k, h)$.

Note that defeat relationship is a refinement of the counterargument relationship; defeaters are only those counterarguments which are at least as strong as the argument being counterargued. Since defeaters are arguments, they can also have defeaters. This leads to a recursive procedure from which a dialectical tree [SCG94] emerges. Formally:

Definition 7. (Dialectical Tree). Let $A$ be an argument for $h$. A dialectical tree for $(A, h)$, denoted $T_{(A, h)}$, is recursively defined as follows:
1) A single node containing an argument $(A, h)$ with no defeaters is by itself a dialectical tree for $(A, h)$.
2) Suppose that $(A, h)$ is an argument with defeaters $(A_1, h_1), \ldots, (A_n, h_n)$ We construct the dialectical tree $T_{(A, h)}$ by putting $(A, h)$ as the root node of it and by making this node the parent node of the roots of the dialectical trees for $(A_1, h_1), (A_2, h_2), \ldots, (A_n, h_n)$.

Nodes in a dialectical tree can be marked as defeated and undefeated according to the following definition.

Definition 8. (Marking a dialectical tree). Nodes in a dialectical tree $T_{(A, h)}$ can be recursively labeled as undefeated nodes (U-nodes) and defeated nodes (D-nodes) as follows: a) Leaves in $T_{(A, h)}$ are U-nodes; b) Let $(B, q)$ be an inner node in $T_{(A, h)}$. Then $(B, q)$ will be a U-node iff every child node of $(B, q)$ is a D-node. $(B, q)$ will be a D-node iff it has at least one U-node as a child node.

If all defeaters for the root node $(A, h)$ are defeated after performing the marking process on $T_{(A, h)}$, then $(A, h)$ is called a justification.

---

4. Specificity establishes a syntactic criterion for comparing conflicting arguments; other partial order among arguments could also be used instead.

5. There are additional technical requirements on dialectical trees in order to consider them acceptable [SCG94]. We will discuss them in connection with pre-trees in section 5.
Definition 9. (Justification). Let $T_{(A,h)}$ be a dialectical tree. If the label of the root of $T_{(A,h)}$ is an U-node, then $(A,h)$ is said to be a justification, or a justified argument. In that case, the hypothesis $h$ can be accepted as justified belief.

The operator not corresponding to “negation as failure” in logic programming has a special semantics in DLP [GS99]: not $q$ succeeds when there is no justification for $q$; not $q$ fails when there is a justification for $q$.

Example 2. Considering the defeasible logic program shown below (left). Then the arguments shown on the right side can be derived.

$$P_1 = \neg p \rightarrow q \quad \langle A, \neg p \rangle, \text{ with } A = \{ \neg p \rightarrow q, q \rightarrow \text{True} \}.$$  
$$p \rightarrow q, r \quad \langle B, p \rangle, \text{ with } B = \{ p \rightarrow q, r, q \rightarrow \text{True}, r \rightarrow \text{True} \}.$$  
$$q \rightarrow \quad \langle C, d \rangle, \text{ with } C = \{ d \rightarrow \neg p, \neg p \rightarrow q, q \rightarrow \text{True} \}.$$  
$$r \rightarrow \quad \langle E, s \rangle, \text{ with } E = \{ s \rightarrow q, not d, q \rightarrow \text{True} \}.$$  

Note that $\langle B, p \rangle$ defeats $\langle A, \neg p \rangle$, and $\langle B, p \rangle$ is undefeated. Hence $\langle B, p \rangle$ is a justification, whereas $\langle A, \neg p \rangle$ is not. Since the argument $\langle E, s \rangle$ involves not $d$, in order to determine whether it is justified or not, it must be tested whether $d$ is justified. There is an argument $\langle C, d \rangle$ supporting $d$, but $\langle C, d \rangle$ is defeated by $\langle B, p \rangle$. Hence $\langle E, s \rangle$ is justified.

4 Recasting DLP into Extended Logic Programs

In [?], a legal reasoning approach is suggested as a generic way of modeling argumentation. Their idea is basically to reduce defeasible and non-defeasible implications into a single form, by adding (if necessary) so-called non-provability claims. According to this approach, defeasible rules of the form $P$ if $Q$ can be rewritten as $P$ if $Q$ and $S$ cannot be shown, where $S$ represents one or more non-provability claims. Hence, statements such as “thieves should be punished” (which correspond to the defeasible implication $\text{punished}(X) \rightarrow \text{thief}(X)$) can be rewritten as

$$\text{punished}(X) \leftarrow \text{thief}(X) \text{ and cannot be shown } \neg \text{punished}(X)$$

Example adapted from [GS99].

This approach has a direct analogy with Reiter’s default rules [Rei80]

$$A : M B \quad B$$

where $M B$ denotes “it is consistent to assume $B$”, or equivalently “$\neg B$ cannot be shown”. The main difference however is that Reiter’s formulation involves inference rules, whereas the proposed approach includes non-provability claims in the object language level.
Program clauses in DLP Rewritten as

| p←¬q       | p←¬q       |
| p←q        | p←q, ¬(¬p) |
| p←¬q       | p←¬q, ¬(¬p) |

Fig. 1. Clauses in DLP rewritten using the *not* operator

For the sake of simplicity, we will write *not* to denote the non-provability operator *cannot be shown*. Kowalski and Toni [?] contend that using this approach argumentation can be expressed as a dialectical process in which the proponent advances an “exact” argument for a conclusion, supported in defeasible non-provability claims, having the form *not* $S$. The opponent can then ‘undermine’ these assertions presenting an argument for $S$, ‘attacking’ *not* $S$. Like the proponent, the opponent is also allowed to introduce other non-provability claims to conclude $S$.

In the MTDR formalism (and consequently in DLP), the intuitive semantics of applying a ground instance of a defeasible rule $B ← a_1, a_2, \ldots, a_n$ when building an argument $\langle A_1, h_1 \rangle$ states that “having reasons to believe $a_1, a_2, \ldots, a_n$ provide reasons to believe $B$”, leaving implicit “although the opposite might also be defeasibly inferred” (when building another argument $\langle A_2, h_2 \rangle$). Hence the complement of the head of a defeasible rule $r$ can be thought of as the non-provability claim associated with $r$, so that such a rule could be rewritten as a “strict” implication of the form

$$b ← a_1, a_2, \ldots, a_n, \neg(\neg b)$$

Note that DLP inherits the *not* operator as “negation as failure” for both strong and defeasible rules, but with a different procedural semantics (*not* $p$ succeeds if $p$ is not justified, and fails otherwise). We suggest that the same non-provability operator *not* associated with recasting defeasible rules into strict implications can be used for expressing the notion of “negation as failure to justify”, as we will show in the next section. Since we want to find analogies between the abstract argumentation approach for logic programming and DLP, we will recast DLP programs into logic programs using extended logic program format.\footnote{The resulting programs would syntactically resemble extended logic programs as defined in [DPP97], although the semantics for *not* would be different to the one associated with traditional “negation as failure”.}

**Definition 10.** (Extended Logic program $E(P)$ based on $P$). Let $P$ be a defeasible logic program. We will write $E(P)$ to denote the extended logic program based on $P$ which can be obtained performing the following steps:

- **Step 1:** For every defeasible clause $B ← A$ in $P$, there is a clause $B ← A, \neg(\neg B)$ in $E(P)$. Every strong clause $B ← A$ in $P$, is also a clause in $E(P)$.\footnote{The resulting programs would syntactically resemble extended logic programs as defined in [DPP97], although the semantics for *not* would be different to the one associated with traditional “negation as failure”.}
Step 2: Replace each occurrence of not by not in $E(P)$.

We assume that strong negation $\sim$ behaves symmetrically. Thus if a literal $\sim(\sim B)$ results, it should be rewritten as $B$.

Figure 1 shows how defeasible program clauses look like when rewritten according to definition 10. Given an extended logic program $P$, the set of hypothesis associated with $P$ will be denoted $\mathcal{H}(P) = \{ \text{not } p \mid p \text{ is a variable-free objective literal in the Herbrand base of } P \}$.

Example 3. Consider example 2. That program can be rewritten as follows.

$$
E(P_1) = \sim p \leftarrow q, \text{not } p \\
p \leftarrow q, r, \text{not}(\sim p) \\
q \leftarrow \text{not}(\sim q) \\
r \leftarrow \text{not}(\sim r) \\
d \leftarrow \sim p, \text{not}(\sim d) \\
s \leftarrow q, \text{not } d, \text{not}(\sim s)
$$

By adapting this formalization, MTDR arguments should be accordingly redefined, since all rules are ‘strong’ rules from now on. Instead of considering an argument $A$ as a subset of defeasible clauses in a defeasible logic program $P$, we will consider an argument to be a subset of clauses $A \subseteq P$, such that there exists a set of hypotheses $\mathcal{H}(A) \subseteq \mathcal{H}(P)$. Conditions concerning derivability, minimality and inconsistency also apply.

We will introduce new definitions for argument, counterargument, and defeat by considering this setting. In order to distinguish these new definitions from the original ones, we will precede them with the prefix $H$.

**Definition 11.** (H-Argument). Let $P$ be an extended logic program, and let $\mathcal{H}(P)$ be its associated set of hypothesis. We will say that a grounded subset $A \subseteq P$ is an $H$-argument for a grounded literal $h$ iff

- $A \cup \mathcal{H}(A) \vdash h$
- it does hold that:
  - $A \cup \mathcal{H}(A) \not\vdash p, \sim p$
  - $A \cup \mathcal{H}(A) \not\vdash p$ if $\text{not } p \in \mathcal{H}(A)$
- $\not\exists A' \subseteq A$ such that $A' \cup \mathcal{H}(A) \vdash h$.

We will write $(A_1, h_1)$ to denote that $A_1$ is an $H$-argument for $h_1$. An $H$-argument $(B, q)$ is an $H$-subargument of $(A, h)$ if $B \subseteq A$.

---

9 In extended logic programming [DPP97], an atom has the form $p(t_1, \ldots, t_n)$, where $p$ is a predicate and $t_i$’s are terms. Objective literals are defined over predicates as being an atom $A$ or its symmetric negation $\sim A$. The symbol $\sim$ is used to denote complementary literals in the sense of symmetric negation. Thus $\sim \sim A = A$. A literal is either an objective literal $L$ or its default negation not $L$.

10 We will overload the notation for arguments when no confusion arises.
Definition 12. (Preference ordering $O_P$). Let $P$ be a logic program, and let $\text{Args}(P)$ denote the set of H-arguments based on $P$. A preference ordering $O_P \subseteq \text{Args}(P) \times \text{Args}(P)$ is any partial ordering on $\text{Args}(P)$.

Definition 13. (H-Counterargument, H-Defeater). Let $(\mathcal{A}_1, h_1)$ and $(\mathcal{A}_2, h_2)$ be two H-arguments. Then we will say that $(\mathcal{A}_2, h_2)$ H-counterargues $(\mathcal{A}_1, h_1)$ iff there exists $(\mathcal{A}, h)$ H-subargument of $(\mathcal{A}_1, h_1)$ such that:

- (Complementary literals): $\mathcal{A}_2 \cup \mathcal{H}(\mathcal{A}_2) \cup \{ h_2, h \} \vdash p, \neg p$
- (Proof against assumption): $\mathcal{A}_2 \cup \mathcal{H}(\mathcal{A}_2) \vdash p$ and $\neg p \in \mathcal{H}(\mathcal{A})$.

If $(\mathcal{A}_2, h_2)$ H-counterargues $(\mathcal{A}_1, h_1)$, and $(\mathcal{A}_2, h_2)$ is better (or unrelated to) $(\mathcal{A}, h)$ by $O_P$, then $(\mathcal{A}_2, h_2)$ is an H-defeater for $(\mathcal{A}_1, h_1)$.

Definition 14. (Dialectical H-tree, H-justification). Let $(\mathcal{A}, h)$ be an H-argument. We will define the concept of dialectical H-tree in a similar way to the concept of dialectical tree (def. 7). We define the marking process of a dialectical H-tree in the same way it is defined in dialectical trees (def. 8). Similarly, we will say that $(\mathcal{A}, h)$ is an H-justification iff it is marked as undefeated in the dialectical H-tree $T_{(\mathcal{A}, h)}$.

Example 4. Adopting this formalization, we can consider the program $E(P_1)$ (example 3) and the following sets of rules:

- $\mathcal{A} = \{ \neg p \leftarrow q, \text{not } p, q \leftarrow \neg \neg \neg q \}$, with $\mathcal{H}(\mathcal{A}) = \{ \text{not}(\neg q), \text{not}p \}$
- $\mathcal{B} = \{ p \leftarrow q, r, \text{not}(\neg p), q \leftarrow \neg \neg \neg q, r \leftarrow \text{not}(\neg r), \text{not}(\neg p) \}$
- $\mathcal{C} = \{ d \leftarrow \neg p, \text{not}(\neg d), \neg p \leftarrow q, \text{not}p, q \leftarrow \text{not}(\neg q) \}$ with $\mathcal{H}(\mathcal{C}) = \{ \text{not}(\neg d), \text{not}(\neg q), \text{not}p \}$
- $\mathcal{E} = \{ s \leftarrow q, \text{not}d, \text{not}(\neg s), q \leftarrow \text{not}(\neg q) \}$, with $\mathcal{H}(\mathcal{E}) = \{ \text{not}d, \text{not}(\neg s), \text{not}(\neg q) \}$

Note that in this case, we have the H-arguments $(\mathcal{A}, \neg p)$, $(\mathcal{B}, p)$, $(\mathcal{C}, d)$ and $(\mathcal{E}, s)$. Note that $(\mathcal{B}, p)$ H-counterargues $(\mathcal{A}, \neg p)$, since $P \cup \mathcal{H}(\mathcal{B}) \vdash p$ and $\text{not } p \in A$.

Similarly, $(\mathcal{A}, \neg p)$ H-counterargues $(\mathcal{B}, p)$, since $P \cup \text{Hypo}(\mathcal{A}) \vdash \neg p$ and $\text{not } \neg p \in B$.

Assume we impose a partial order $O_{A(P_1)} = \{ (\mathcal{H}_B, \mathcal{H}_A), (\mathcal{H}_B, \mathcal{H}_C), (\mathcal{H}_C, \mathcal{H}_E) \}$, where $(\mathcal{A}, \mathcal{B})$ denotes “$\mathcal{A}$ defeats $\mathcal{B}$”. This partial order will be for this example equivalent to the one determined by specificity [SL92]. It follows that $(\mathcal{A}, \neg p)$ is an H-justification, since $(\mathcal{B}, p)$ H-defeats $(\mathcal{A}, \neg p)$, $(\mathcal{B}, p)$ H-defeats $(\mathcal{C}, d)$, and $(\mathcal{C}, d)$ H-defeats $(\mathcal{E}, s)$, so that an H-dialectical tree (such as the one discussed in example 2) can be built whose root would be labelled $U$.

This alternative formalization using ELP format highlights some interesting features in DLP, namely:

\footnote{In the case of specificity as preference criterion (as in [SL92]), only set of hypotheses should be considered, but also the set $\mathcal{A}$ of rules used in an argument. This analysis, however, is beyond the scope of this paper.}
Arguments can be thought of as set of program clauses supported by hypotheses, which represent the impossibility of proving the contrary of consequents of defeasible rules.

The use of not for denoting "negation as failure to justify" (as used in DLP) can be understood as an hypothesis (this situation arises in example 4 when \(\langle C, d \rangle\) H-defeats \(\langle E, s \rangle\)).

Defeat in DLP can arise by attacking hypotheses as well as complementary literals (shown in example 5), whereas in normal programs only hypotheses can be attacked.

5 Comparing DLP and Normal Programs

Next we will relate and compare different concepts in the wf-semantics for normal logic programs (under the AFF setting) with those arising in defeasible logic programs under our extended-logic-programming formulation. First, let us consider the notion of well-founded acceptability as stated in def. 2 within our formulation. The only difference is that instead of using attack among arguments we consider H-defeat in terms of a preference criterion \(O_{Args}\).

**Definition 15.** (Well-founded acceptability wrt \(O_{Args}\)). A set of hypothesis \(A\) is well-founded acceptable wrt a preference ordering \(O_{Args}\) (\(wf-O_{Args}\)-acceptable) iff for all sets of hypothesis \(B\), if \(B\) defeats \(A\) on the basis of \(O_{Args}\), then there exists a set \(C\) such that \(C\) defeats \(B\), and \(C\) is \(wf-O_{Args}\)-acceptable.

![Dialectical tree subsuming a well-founded pre-tree](image)

The base case in this recursive definition is the following: a set \(\Delta\) is \(wf-O_{Args}\)-acceptable iff no set of hypotheses \(A\) defeats \(\Delta\). Note that justification (def. 9) can also be expressed in terms of well-founded acceptability (an argument with no defeaters is justified; an argument which has defeaters such that they
are on its turn defeated by other justified arguments is certainly justified). This prompts the following proposition.\footnote{The proof is not shown for reasons of space (for details the reader is referred to \cite{?}).}

**Proposition 1.** Let \( \langle A, h \rangle \) be an H-argument, and let \( T_{A,h} \) be its associated dialectical H-tree. Then \( \langle A, h \rangle \) is an H-justification if \( A \) is \( w.f.O_{A_{\text{Args}}} \)-acceptable.

It should be noted that the converse does not hold; the conflict between arguments may involve not only hypotheses, but also complements of literals. Consider the following example:

**Example 5.** Consider the program \( P = \{ z \leftarrow p, p \leftarrow a, \not \not (\neg p), \neg z \leftarrow b, \not \not (\neg z), a \leftarrow true, b \leftarrow true \} \). Note that the argument \( A = \{ z \leftarrow p, p \leftarrow a, \not \not (\neg p) \} \) with \( H(A) = \{ \not \not (\neg p) \} \) is counterargued by \( B = \{ \neg z \leftarrow b, \not \not z \} \), with \( H(B) = \{ \not \not z \} \). There is no conflict between hypothesis, but between complementary literals \( \neg z \) and \( z \).

Next we will discuss how the notion of well-founded pre-tree (def. 3) relates to H-dialectical trees (def. 14). In a certain sense, every H-dialectical tree whose root is a justified argument ‘entails’ a well-founded pre-tree. In fact, the hypotheses associated to the root \( \langle A, h \rangle \) of an H-dialectical tree account for a “defense node” in a pre-tree, whereas H-defeaters \( \langle B_1, h_1 \rangle \ldots \langle B_k, h_k \rangle \) for \( \langle A, h \rangle \) represent “attack nodes”. Since every H-defeater for \( \langle A, h \rangle \) is defeated (since \( \langle A, h \rangle \) is justified), it is clear that there exists at least one H-defeater \( \langle C, r \rangle \) for every argument (node) \( \langle B_i, h_i \rangle \). That H-defeater is namely the “exactly one child” referred to in def. 3, which allows the reinstatement of the root. This analysis propagates recursively downwards in the dialectical tree, Figure 7 shows in dotted lines the first levels of a well-founded pre-tree subsumed in a dialectical tree with justified root. Two important differences must be remarked, however: 1) dialectical trees involve only attacks on hypotheses, but also on complementary literals; 2) defeat in dialectical trees is based on a preference criterion, whereas attack in pre-trees is not.

Finally, it is important to point out that two technical requirements are needed in dialectical trees in order to consider them acceptable \cite{SCG94}. One of them is demanding \emph{no circularity} (no argument can be used twice when building a branch in a dialectical tree). This requirement is embedded in the definition of well-founded pre-tree (all of its branches are finite). The second requirement involves \emph{consistency} (“defense” nodes should be consistent among themselves within a branch in a dialectical tree; the same applies for “attack” nodes). In pre-trees, a similar situation arises from the definition: all “defense” nodes in a pre-tree will be consistent among themselves.\footnote{This has been stated as a theorem in \cite{KT99}.}

## 6 Conclusions

We have outlined a generalized way of performing defeasible argumentation with extended logic programming. We show how defeasible rules can be embedded as
Normal Logic Programming (under AAF setting) | Defeasible Logic Programming (under ELP, AAF-like setting)
--- | ---
An argument is a set of hypotheses $\Delta$ satisfying derivability | An argument is a set of hypothesis $H(\Delta)$ satisfying consistency, minimality, derivability
Attacks defined on hypotheses $p$ attacks not $p$ | Attack defined on complementary literals $p$ and $\neg p$, and hypotheses $p$ and not $p$
Attacks involve no preference | Attacks involves preference
Well-founded semantics | Well-founded acceptability restricted by dialectical considerations and preference criterion
Dialectical considerations:
- no circularity
- consistency among all defense nodes | Dialectical considerations:
- no circularity
- consistency within defense and attack nodes within argumentation lines
Existence of well-founded pre-tree iff well-founded acceptability | Existence of a dialectical tree which entails pre-tree structures Root marked $U$ iff justification

**Fig. 3.** Normal Logic Programming and DLP under AAF settings

strict rules by adding appropriate non-provability claims. This allows us to combine negation as “failure to justify” together with the notion of defeat, unifying these two concepts.

Our primary goal was to find relationships between well-founded semantics in normal programs and the semantics of defeasible logic programs. Our results are summarized in figure 3. Defeasible logic programming is richer than normal logic programs for knowledge representation: preference between conflicting arguments can be solved, and two kinds of negation are introduced. However, our analysis showed that the corresponding semantics are structurally similar.

Defeasible logic programming imposes more requirements on a set of hypotheses to be accepted as part of a justified argument. Further research should be focused on defining a justification semantics, which seems to be a restricted version of well-founded semantics by considering a preference criterion on arguments. Work in this direction is being currently pursued.

**Acknowledgments:** The authors wish to thank anonymous referees for comments and suggestions which helped improve the final version of this paper. Carlos Iván Chesñevar wishes to thank Francesca Toni (Imperial College, London, UK) for discussions about their abstract argumentation framework in comparison with the MTDR approach.

**References**


