Relating Defeasible and Normal Logic Programming through Transformation Properties

Carlos Iván Chesñevar, cic@cs.uns.edu.ar
Jürgen Dix, dix@uni-koblenz.de
Frieder Stolzenburg, stolzen@uni-koblenz.de
Guillermo Ricardo Simari, grs@cs.uns.edu.ar

1 Universidad Nacional del Sur, Av. Alem 1253, (8000) Bahía Blanca, ARGENTINA
2 Universität Koblenz-Landau, Rheinau 1, 56075 Koblenz, GERMANY

Abstract. This paper relates the Defeasible Logic Programming (DeLP) framework and its semantics SEM_{DeLP} to more classical logic programming frameworks. In DeLP we distinguish between strict and defeasible rules, combining default and strict negation. In contrast to this, in normal logic programming (NLP), there is one negation not, interpreted as a kind of negation-as-failure, which introduces defeasibility. Various semantics have been defined for NLP, notably the well-founded semantics WFS.

In this paper we consider the transformation properties for NLP introduced by Brass et al. adapted within the DeLP framework. We show which transformation properties are satisfied, identifying the aspects in which NLP and DeLP differ. We contend that transformation rules presented in this paper can help to gain a better understanding of the relationship of DeLP semantics with respect to more traditional logic programming approaches. As a byproduct we get that DeLP is a proper extension of NLP.

Key words: defeasible argumentation; knowledge representation; logic programming; non-monotonic reasoning.

1 Introduction and motivations

Defeasible Logic Programming (DeLP) is a logic programming formalism which relies upon defeasible argumentation for solving queries. Logic programming has experienced considerable growth in the last decade, and several extensions have been developed and studied, such as normal logic programming (NLP) and extended logic programming (ENLP). For these formalizations different semantics have been developed, such as well-founded semantics (WFS) and stable model semantics. In contrast, DeLP has an ‘operational’ semantics which is determined by the outcome of the dialectical process used for answering queries.

In [BD99], a number of transformation rules were introduced which allow to ‘simplify’ a normal logic program (NLP) P to get its WFS. The application of these rules leads to a new, simplified NLP P' from which its WFS can be easily read off. In this paper we will focus on finding similar transformation rules for DeLP, which can be used to simplify the knowledge encoded in a DeLP program. In our analysis, we show that in DeLP a complete simplification of the original program cannot be achieved. However, our results suggest some connections between the semantics of classical approaches and logic programming with DeLP.

The paper is structured as follows: Section 2 introduces preliminary notions concerning NLP and DeLP. Section 3 introduces transformations for NLP. Section 4 shows how to adapt these transformations for DeLP. Section 5 summarizes the relationships between NLP and DeLP, and the main results we have obtained. Finally, Section 6 concludes.

* This paper emerged while the first author was visiting the University of Koblenz in February 2000. It is part of the joint Argentine-German collaboration project DeReLoP.

1 See [DPP97,BD01] for an in-depth discussion of extensions of logic programming and their semantics.
2 Preliminaries

In order to render the paper in a self-contained manner, this section contains all the necessary definitions. Subsection 2.1 introduces normal logic programs, and Subsection 2.2 introduces the defeasible logic programming framework. We will focus our analysis on propositional logic programs.\(^2\)

2.1 Normal Logic Programs (NLP)

**Definition 2.1 (Normal Logic Program \(\mathcal{P}\)).** A normal logic program (nlp) \(\mathcal{P}\) is a finite set of normal program rules. A normal program rule has the form \(A \leftarrow L_1, \ldots, L_n\), where \(A\) is an atom and each \(L_i\) is an atom \(B\) or its negation \(\neg B\). If \(B = \{L_1, \ldots, L_n\}\) is the body of a rule \(A \leftarrow B\), we also use the notation \(A \leftarrow B^+, \neg B^-\), where \(B^+\) (resp. \(B^-\)) contains all the positive (resp. negative) body atoms in \(B\).

In NLP, atoms \(A\) and negated atoms \(\neg A\) are called *literals*. However, we must not confuse this notion with the notion of a literal introduced in Section 2.2. In the sequel we will speak of an atom and its negation, referring to an atom \(A\) and its default negation \(\neg A\). If \(B^+ = B^- = \emptyset\), we say that the rule is a fact and denote it by \(A \leftarrow \) (or just by \(A\)).

We will now introduce some concepts useful for describing what a semantics of a nlp is. Let \(\text{Prog}_\mathcal{L}\) be the set of all normal propositional programs with atoms from a signature \(\mathcal{L}\). By \(\mathcal{L}_\mathcal{P}\) we understand the signature of \(\mathcal{P}\), i.e. the set of atoms that occur in \(\mathcal{P}\). A (partial) interpretation based on a signature \(\mathcal{L}\) is a disjoint pair of sets \(\langle I_1, I_2 \rangle\) such that \(I_1 \cup I_2 \subseteq \mathcal{L}\). A partial interpretation is total if \(I_1 \cup I_2 = \mathcal{L}\). We may also view an interpretation \(\langle I_1, I_2 \rangle\) as the set of atoms and negated atoms \(I_1 \cup \neg I_2\).

**Definition 2.2 (Semantics SEM).** A semantics \(\text{SEM}\) is a mapping which assigns to every logic program \(\mathcal{P}\) a set \(\text{SEM}(\mathcal{P})\) of (partial) models of \(\mathcal{P}\), such that \(\text{SEM}\) is “instantiation invariant”, i.e. \(\text{SEM}(\mathcal{P}) = \text{SEM}(\text{ground}(\mathcal{P}))\), where \(\text{ground}(\mathcal{P})\) denotes the Herbrand instantiation of \(\mathcal{P}\). A semantics \(\text{SEM}\) is called 3-value based if for every program \(\mathcal{P}\) the partial interpretation \(\text{SEM}(\mathcal{P})\) is a 3-valued model\(^3\) of \(\mathcal{P}\).

In Section 3 we will consider a particular 3-valued semantics for nlp called WFS, which can be computed by applying transformation rules on a nlp \(\mathcal{P}\).

2.2 Defeasible Logic Programs (DeLP)

The DeLP language [SL92,Gar97,GSC98] is defined in terms of two disjoint sets of rules: a set of strict rules for representing strict (sound) knowledge, and a set of defeasible rules for representing tentative information. Rules will be defined using literals. A literal \(L\) is an atom \(p\) or a negated atom \(\neg p\), where the symbol “\(\neg\)” represents strong negation. We define this formally:

**Definition 2.3 (Strict, \(\leftarrow\), and Defeasible Rules, \(\leftarrow\)).** A strict rule (defeasible rule) is an ordered pair, conveniently denoted by \(\text{Head} \leftarrow \text{Body}\) (\(\text{Head} \leftarrow \neg \text{Body}\)), whose first member, \(\text{Head}\), is a literal, and whose second member, \(\text{Body}\), is a finite set of literals. A strict rule (defeasible rule) with the head \(L_0\) and body \(\{L_1, \ldots, L_n\}\) can also be written as \(L_0 \leftarrow L_1, \ldots, L_n\) (\(L_0 \leftarrow \neg L_1, \ldots, \neg L_n\)). If the body is empty, it is written \(L \leftarrow \text{true}\) (\(\neg L \leftarrow \text{true}\)), and it is called a fact (presumption). Facts may also be written as \(L\).

\(^2\) Following [Li94], program rules with variables are viewed as “schemata” that represent their ground instances.

\(^3\) We equip \(\leftarrow\) with the Kleene interpretation, where \(\text{undef} \leftarrow \text{undef}\) is considered to be true.
In the sequel, atoms will be denoted with lowercase letters \( (a, b, \ldots) \). The letter \( r \) (possibly subindicated) will be used for denoting rule names. Literals (i.e. an atom or a negated atom) will be denoted with capital letters \( (A, B, \ldots) \), possibly subindicated. Sets will be denoted as \( \mathcal{A}, \mathcal{B}, \ldots \), possibly subindicated. Logic programs will be usually denoted as \( \mathcal{P}_1, \mathcal{P}_2, \text{ etc.} \)

**Definition 2.4 (Defeasible Logic Program \( \mathcal{P} \)).** A defeasible logic program \( dlp \) is a finite set of strict and defeasible rules. If \( \mathcal{P} \) is a dlp, we will distinguish in \( \mathcal{P} \) the subset \( \Pi \) of strict rules, and the subset \( \Delta \) of defeasible rules. When required, we will denote \( \mathcal{P} \) as \( (\Pi, \Delta) \). We will distinguish the class of all defeasible logic programs that use only strict (resp. default) negation, denoting them as \( \text{DeLP}_{\text{neg}} \) (\( \text{DeLP}_{\text{not}}, \text{ resp.} \)).

Given a \( dlp \), a defeasible derivation for a query \( Q \) is a finite set of rules obtained by backward chaining from \( Q \) as in a Prolog program, using both strict and defeasible rules from the given \( dlp \). The symbol \( \sim \) is considered as part of the predicate when generating a defeasible derivation. A set of rules \( S \) is contradictory iff there is a defeasible derivation from \( S \) for some literal \( P \) and its complement \( \sim P \). Given a \( dlp \), we will assume that the set \( \Pi \) of strict rules is non-contradictory.\(^4\)

**Definition 2.5 (Defeasible Derivation Tree).** Let \( \mathcal{P} \) be a dlp, and let \( H \) be a ground literal. A defeasible derivation tree \( T \) for \( H \) is a finite tree, where all nodes are labelled with literals, satisfying the following conditions:

1. The root node of \( T \) is labelled with \( H \).
2. For each node \( N \) in \( T \) labelled with the literal \( L \), there exists a ground instance of a strict or defeasible rule \( r \in \mathcal{P} \) with head \( L_0 \) and body \( \{L_1, L_2, \ldots, L_k\} \) in \( \mathcal{P} \), such that \( L = L_0 \) for some ground variable substitution \( \sigma \), and the node \( N \) has exactly \( k \) children nodes labelled as \( L_1\sigma, L_2\sigma, \ldots, L_k\sigma \).

The sequence \( S = [r_1, r_2, \ldots, r_k] \) of grounded instances of strict and defeasible rules used in building \( T \) will be called a defeasible derivation of \( H \).

**Definition 2.6 (Argument/Subargument).** Given a \( dlp \), an argument \( \mathcal{A} \) for a query \( Q \), denoted \( \langle \mathcal{A}, Q \rangle \), is a subset of ground instances of the defeasible rules of \( \mathcal{P} \), such that:

1. there exists a defeasible derivation for \( Q \) from \( \Pi \cup \mathcal{A} \),
2. \( \Pi \cup \mathcal{A} \) is non-contradictory; and
3. \( \mathcal{A} \) is minimal with respect to set inclusion.

An argument \( \langle \mathcal{A}_1, Q_1 \rangle \) is a sub-argument of another argument \( \langle \mathcal{A}_2, Q_2 \rangle \), if \( \mathcal{A}_1 \subseteq \mathcal{A}_2 \).

Given a \( dlp \) program \( \mathcal{P} \), we will denote by \( \text{Args}(\mathcal{P}) \) the set of all possible arguments that can be built from \( \mathcal{P} \).

**Definition 2.7 (Counterargument).** An argument \( \langle \mathcal{A}_1, Q_1 \rangle \) counterargues an argument \( \langle \mathcal{A}_2, Q_2 \rangle \) at a literal \( Q \) iff there is a subargument \( \langle \mathcal{A}, Q \rangle \) of \( \langle \mathcal{A}_2, Q_2 \rangle \) such that the set \( \Pi \cup \{Q_1, Q\} \) is contradictory.

Informally, a query \( Q \) will succeed if the supporting argument is not defeated; that argument becomes a justification. In order to establish if \( \mathcal{A} \) is a non-defeated argument, counterarguments that could be defeaters for \( \mathcal{A} \) are considered, i.e. counterarguments that are preferred to \( \mathcal{A} \) according to some criterion. \( \text{DeLP} \) considers a particular preference criterion called specificity [SL92,GSC98] which favors an argument with greater information content and/or less use of defeasible rules.\(^5\)

\(^4\) If a contradictory set of strict rules is used in a \( dlp \) the same problems as in extended logic programming would appear. The corresponding analysis has been done elsewhere [GL90].

\(^5\) See [GSC98] for details.
Definition 2.8 (Proper Defeater / Blocking Defeater). An argument \( \langle A_1, Q_1 \rangle \) defeats \( \langle A_2, Q_2 \rangle \) at a literal \( Q \) iff there exists a subargument \( \langle A, Q \rangle \) of \( \langle A_2, Q_2 \rangle \) such that \( \langle A_1, Q_1 \rangle \) counterargues \( \langle A_2, Q_2 \rangle \) at \( Q \), and either: (a) \( \langle A_1, Q_1 \rangle \) is “better” that \( \langle A, Q \rangle \) (then \( \langle A_1, Q_1 \rangle \) is a proper defeater of \( \langle A, Q \rangle \)); or (b) \( \langle A_1, Q_1 \rangle \) is unrelated by the preference order to \( \langle A, Q \rangle \) (then \( \langle A_1, Q_1 \rangle \) is a blocking defeater of \( \langle A, Q \rangle \)).

Since defeaters are arguments, there may exist defeaters for the defeaters and so on. That prompts for a complete dialectical analysis to determine which arguments are ultimately defeated. Ultimately undefeated arguments will be marked as \( U \)-nodes, and the defeated ones as \( D \)-nodes. Next, we state the formal definitions required for this process:

Definition 2.9 (Dialectical Tree). Let \( A \) be an argument for \( Q \). A dialectical tree for \( \langle A, Q \rangle \), denoted \( T_{\langle A, Q \rangle} \), is recursively defined as follows:

1. A single node labeled with an argument \( \langle A, Q \rangle \) with no defeaters (proper or blocking) is by itself the dialectical tree for \( \langle A, Q \rangle \).
2. Let \( \langle A_1, Q_1 \rangle, \langle A_2, Q_2 \rangle, \ldots, \langle A_n, Q_n \rangle \) be all the defeaters (proper or blocking) for \( \langle A, Q \rangle \). We construct the dialectical tree for \( \langle A, Q \rangle \), \( T_{\langle A, Q \rangle} \), by labeling the root node with \( \langle A, Q \rangle \) and by making this node the parent node of the roots of the dialectical trees for \( \langle A_1, Q_1 \rangle, \langle A_2, Q_2 \rangle, \ldots, \langle A_n, Q_n \rangle \).

Definition 2.10 (Marking of the Dialectical Tree). Let \( \langle A, Q \rangle \) be an argument and \( T_{\langle A, Q \rangle} \) its dialectical tree, then:

1. All the leaves in \( T_{\langle A, Q \rangle} \) are marked as \( U \)-nodes.
2. Let \( \langle B, H \rangle \) be an inner node of \( T_{\langle A, Q \rangle} \). Then \( \langle B, H \rangle \) will be a \( U \)-node iff every child of \( \langle B, H \rangle \) is a \( D \)-node. The node \( \langle B, H \rangle \) will be a \( D \)-node iff it has at least a child marked as \( U \)-node.

To avoid the occurrence of fallacious argumentation [SCG94], some additional constraints on dialectical trees are imposed, giving rise to acceptable dialectical trees. An argument \( A \) which turns to be ultimately undefeated is called a justification. Formally:

Definition 2.11 (Justification). Let \( A \) be an argument for a literal \( Q \), and let \( T_{\langle A, Q \rangle} \) be its associated acceptable dialectical tree. The argument \( A \) for \( Q \) will be a justification iff the root of \( T_{\langle A, Q \rangle} \) is a \( U \)-node.

A given query \( Q \) can be associated with a particular answer set according to some criterion. Several criteria have been analyzed corresponding to different outcomes in the dialectical process. A possible criterion is specified in the following definition [Gar97]:

Definition 2.12 (Answers to a Given Query \( Q \)). Given a dlp \( P \), a query \( Q \) can be classified as a positive, negative, undecided or unknown answer as follows:

1. \( Q \) is a positive answer iff there exists a justification \( \langle A, Q \rangle \).
2. \( Q \) is a negative answer iff for every argument \( \langle A, Q \rangle \), in the dialectical tree \( T_{\langle A, Q \rangle} \), there exists at least a proper defeater for \( A \) marked as \( U \).
3. \( Q \) is an undecided answer iff \( Q \) is not justified, and for every argument \( \langle A, Q \rangle \), it is the case that \( T_{\langle A, Q \rangle} \) has at least one blocking defeater marked as \( U \).
4. \( Q \) is an unknown answer if there is no argument for \( Q \).

---

6 For space reasons we do not discuss these conditions in this paper; see [GSC98] for an in-depth analysis.
Given a dlp $\mathcal{P}$, we call \text{Positive($\mathcal{P}$)}, \text{Negative($\mathcal{P}$)}, \text{Undefined($\mathcal{P}$)} and \text{Unknown($\mathcal{P}$)} the sets of positive, negative, undecided and unknown answers, resp.

From the previous definition we can give a 3-valued semantics $\text{SEM}_{\text{DelP}}(\mathcal{P})$ for a dlp $\mathcal{P}$, classifying literals in $\mathcal{P}$ as accepted, rejected or undefined as follows:

**Definition 2.13 (SEM\textsubscript{DelP}).** For any dlp $\mathcal{P}$, we define $\text{SEM}_{\text{DelP}}(\mathcal{P}) = \langle \mathcal{P}^{\text{accepted}}, \mathcal{P}^{\text{rejected}}, \mathcal{P}^{\text{undefined}} \rangle$, where

\[
\mathcal{P}^{\text{accepted}} = \{ Q | Q \in \text{Justified}(\mathcal{P}) \}, \\
\mathcal{P}^{\text{rejected}} = \{ Q | Q \in \text{Unknown}(\mathcal{P}) \cup \text{Negative}(\mathcal{P}) \}, \\
\mathcal{P}^{\text{undefined}} = \{ Q | Q \in \text{Undefined}(\mathcal{P}) \}.
\]

It must be remarked that since the semantics of DelP is entirely determined by relationships among arguments, two dlp programs $\mathcal{P}$ and $\mathcal{P}'$ would have the same semantics iff $\text{Args}(\mathcal{P}) = \text{Args}(\mathcal{P}')$. This equivalence will be frequently used in the following sections.

3 Transformations for NLP: classifying well-founded semantics

A program transformation is a relation $\leftrightarrow$ between ground logic programs [BDFZ01]. A semantics SEM allows a transformation $\leftrightarrow$ iff $\text{SEM}(\mathcal{P}_1) = \text{SEM}(\mathcal{P}_2)$, for all $\mathcal{P}_1$ and $\mathcal{P}_2$, such that $\mathcal{P}_1 \leftrightarrow \mathcal{P}_2$. In this case we also say that the transformation $\leftrightarrow$ holds wrt SEM. Well-founded semantics for NLP can be elegantly characterized by a set of transformation rules [BD99], which reduce a given nlp program $\mathcal{P}$ into a simplified version $\mathcal{P}'$, from which the WFS can be easily read off.

**Definition 3.1 (Transformation Rules for WFS).** Given a program $\mathcal{P} \in \text{Prog}_G$, let $\text{HEAD}(\mathcal{P})$ be the set of all head-atoms of $\mathcal{P}$, i.e. $\text{HEAD}(\mathcal{P}) = \{ H | H \leftarrow B^+, \text{not } B^- \in \mathcal{P} \}$. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be ground programs. The following transformation rules characterize WFS:

- **RED\textsuperscript{+}: (Positive Reduction)** Program $\mathcal{P}_2$ results from program $\mathcal{P}_1$ by RED\textsuperscript{+} (written $\mathcal{P}_1 \rightarrow \mathcal{P}_2$) iff there is a rule $H \leftarrow B$ in $\mathcal{P}_1$ and a negative literal $\text{not } B \in B$ such that there is no rule about $B$ in $\mathcal{P}_1$, i.e. $B \notin \text{HEAD}(\mathcal{P}_1)$, and $\mathcal{P}_2 = (\mathcal{P}_1 - \{ H \leftarrow B \}) \cup \{ H \leftarrow (B - \text{not } B) \}$.

- **RED\textsuperscript{-}: (Negative Reduction)** Program $\mathcal{P}_2$ results from program $\mathcal{P}_1$ by RED\textsuperscript{-} (written $\mathcal{P}_1 \rightarrow \mathcal{P}_2$) iff there is a rule $H \leftarrow B$ in $\mathcal{P}_1$ and a negative literal $\text{not } B \in B$ such that $B$ appears as a fact in $\mathcal{P}_1$, and $\mathcal{P}_2 = \mathcal{P}_1 - \{ H \leftarrow B \}$.

- **SUB: (Deletion of non-minimal rules)** Program $\mathcal{P}_2$ results from program $\mathcal{P}_1$ by SUB (written $\mathcal{P}_1 \rightarrow \mathcal{P}_2$) iff there are rules $H \leftarrow B$ and $H \leftarrow B'$ in $\mathcal{P}_1$ such that $B \subseteq B'$ and $\mathcal{P}_2 = \mathcal{P}_1 - \{ H \leftarrow B' \}$.

- **UNFOLD: (Unfolding)** Program $\mathcal{P}_2$ results from program $\mathcal{P}_1$ by UNFOLD (written $\mathcal{P}_1 \rightarrow \mathcal{P}_2$) iff there is a rule $H \leftarrow B$ in $\mathcal{P}_1$ and a positive literal $B \in B$ such that $\mathcal{P}_2 = \mathcal{P}_1 - \{ H \leftarrow B \}$ or $\mathcal{P}_2 = \mathcal{P}_1 - \{ H \leftarrow (B - \text{not } B) \}$. 

- **TAUT: (Deletion of Tautologies)** Program $\mathcal{P}_2$ results from program $\mathcal{P}_1$ by TAUT (written $\mathcal{P}_1 \rightarrow \mathcal{P}_2$) iff there is $H \leftarrow B \in \mathcal{P}_1$ such that $H \in B$ and $\mathcal{P}_2 = \mathcal{P}_1 - \{ H \leftarrow B \}$.

A program $\mathcal{P}'$ is a normal form of a program $\mathcal{P}$ wrt a transformation “$\rightarrow$” iff $\mathcal{P} \rightarrow \mathcal{P}'$, and $\mathcal{P}'$ is irreducible, i.e. there is no program $\mathcal{P}''$ such that $\mathcal{P}' \rightarrow \mathcal{P}''$. 

Let “\(\rightarrow_R\)” be the rewriting system consisting of the above five transformations, i.e. \(\rightarrow_R = \rightarrow_T \cup \rightarrow_U \cup \rightarrow_M \cup \rightarrow_P \cup \rightarrow_N\). Two distinctive features of this rewriting system [BD98] are that it is terminating (i.e. every ground program \(\mathcal{P}\) has a normal form \(\mathcal{P}'\)), and confluent (i.e. given a program \(\mathcal{P}\), by applying the transformations in any order, we eventually arrive at a normal form \(\text{norm}_{\text{WFS}}(\mathcal{P})\)). This normal form \(\text{norm}_{\text{WFS}}(\mathcal{P})\) is a residual program, consisting of rules without positive body atoms. For such a simplified program, its well-founded semantics can be easily read off as follows:

**Definition 3.2 (SEM\(_{\text{min}}\)).** For any nlp \(\mathcal{P}\), we define \(\text{SEM}_{\text{min}}(\mathcal{P}) = (\mathcal{P}_{\text{true}}, \mathcal{P}_{\text{false}}, \mathcal{P}_{\text{undef}})\), where

\[
\begin{align*}
\mathcal{P}_{\text{true}} &= \{ H \mid H \leftarrow \in \mathcal{P} \} \\
\mathcal{P}_{\text{false}} &= \{ H \mid H \in \mathcal{L}_P - \text{HEAD}(\mathcal{P}) \} \\
\mathcal{P}_{\text{undef}} &= \{ H \mid H \in \mathcal{L}_P - (\mathcal{P}_{\text{true}} \cup \mathcal{P}_{\text{false}}) \}
\end{align*}
\]

**Theorem 3.3 (Classifying WFS [BD99]).** \(WFS(\mathcal{P}) = \text{SEM}_{\text{min}}(\text{norm}_{\text{WFS}}(\mathcal{P}))\).

### 4 Transformation Properties in DeLP

As stated in the introduction, we want to analyze whether transformations for NLP as the ones described above also hold for a DeLP program. In our analysis, we will focus first on DeLP\(_{\text{neg}}\) (i.e., DeLP with strict negation “\(\sim\)”). As the transformations in [BDFO01] are defined with respect to a NLP setting, we will adapt them accordingly. Therefore, we extend our previous terminology to be applied to a DeLP\(_{\text{neg}}\) program \(\mathcal{P}\) (thus \(\text{HEAD}(\mathcal{P})\) will stand for all heads of rules in \(\mathcal{P}\), etc.), distinguishing strict rules from defeasible rules when needed. Next, in section 4.2 we will consider DeLP\(_{\text{not}}\) (i.e., DeLP with default negation “\(\text{not}\)”). In that case, a similar analysis will be performed.

#### 4.1 Transformation Properties in DeLP\(_{\text{neg}}\)

Below we will introduce tentative extensions to DeLP\(_{\text{neg}}\) of the previous transformation rules. The distinguishing features of the transformation rules are discussed next. For every transformation, \(\mathcal{P}_1\) and \(\mathcal{P}_2\) denote ground dlp programs. Some transformation rules have special requirements which appear underlined.

**RED\(_{\text{neg}}^+\):** Program \(\mathcal{P}_2\) will result from program \(\mathcal{P}_1\) by RED\(_{\text{neg}}^+\) (written \(\mathcal{P}_1 \rightarrow_{\text{neg}^+} \mathcal{P}_2\)) iff there is a rule \(H \leftarrow B\) in \(\mathcal{P}_1\) and a negative literal \(\sim B \in B\) such that there is no rule about \(B\) in \(\mathcal{P}_1\), i.e. \(B \notin \text{HEAD}(\mathcal{P}_1)\), and \(\mathcal{P}_2 = (\mathcal{P}_1 - \{H \leftarrow B\}) \cup \{H \leftarrow (B - \{\sim B\})\}\).

**RED\(_{\text{neg}}^-\):** Program \(\mathcal{P}_2\) will result from program \(\mathcal{P}_1\) by RED\(_{\text{neg}}^-\) (written \(\mathcal{P}_1 \rightarrow_{\text{neg}^-} \mathcal{P}_2\)) iff there is a rule \(H \leftarrow B\) in \(\mathcal{P}_1\) and a negative literal \(\sim B \in B\) such that \(B\) appears as a fact in \(\mathcal{P}_1\), and \(\mathcal{P}_2 = \mathcal{P}_1 - \{H \leftarrow B\}\).

**SUB\(_{\text{neg}}\):** Program \(\mathcal{P}_2\) will result from program \(\mathcal{P}_1\) by SUB (written \(\mathcal{P}_1 \rightarrow_M \mathcal{P}_2\)) iff there are strict rules \(H \leftarrow B\) and \(H \leftarrow B'\) in \(\mathcal{P}_1\) such that \(B \subset B'\) and \(\mathcal{P}_2 = \mathcal{P}_1 - \{H \leftarrow B\}\). The rule \(H \leftarrow B_2\) is called non-minimal rule wrt \(H \leftarrow B_1\).

**UNFOLD\(_{\text{neg}}\):** Suppose program \(\mathcal{P}_1\) contains a strict rule \(H \leftarrow B\) such that there is no defeasible rule in \(\mathcal{P}_1\) with head \(H\). Then program \(\mathcal{P}_2\) will result from program \(\mathcal{P}_1\) by UNFOLD\(_{\text{neg}}\) (written \(\mathcal{P}_1 \rightarrow_{\text{neg}} \mathcal{P}_2\)) iff there is a positive literal \(B \in B_1\) such that

\[\text{UNFOLD}_{\text{neg}}\]

\[\text{Note that we do not distinguish between atoms and their negations because negated literals are treated as new predicate names.}\]
that \( P_2 = P_1 - \{ H \leftarrow B \} \cup \{ H \leftarrow ((B - \{ B \}) \cup B') | B \leftarrow B' \in P_1 \} \). The clause \( H \leftarrow B \) is said to be \( UNFOLD_{neg-related} \) with each \( B \leftarrow B_i \in P_1 \) for \( i = 1, \ldots, n \).

**TAUT**\( _{neg} \): Program \( P_2 \) will result from program \( P_1 \) by **TAUT**\( _{neg} \) (written \( P_1 \leftarrow_{T neg} P_2 \)) iff there is \( H \leftarrow B \in P_1 \) such that \( H \in B \) and \( P_2 = P_1 - \{ H \leftarrow B \} \).

First we consider **RED**\( _{neg}^+ \). This transformation rule does not hold for strict negation. Note that whereas **RED**\( _{neg}^+ \) captures the idea that not \( A \) trivially holds whenever \( A \) cannot be derived (and for that reason not \( A \) can be deleted), the same principle cannot be applied to \( \sim A \), which holds whenever there is a derivation for \( \sim A \). Consider the following example:

**Example 4.1.** Consider the following **DEL**\( _{neg} \) program: \( \Pi = \{ (p \leftarrow \sim s), (s \leftarrow t), (q_1 \leftarrow), (q_2 \leftarrow) \} \) and \( \Delta = \{ (t \leftarrow < q_1), (\sim t \leftarrow < q_1, q_2) \} \). Here \( p \) is not justified from \( P \) (since the argument \( A_1 = \{ t \leftarrow < q_1 \} \) for \( p \) is defeated by the argument \( A_2 = \{ \sim t \leftarrow < q_1, q_2 \} \) for \( \sim t \)). If we consider \( P' = \text{RED}^+_{neg}(P) \) we get \( p \) as a fact, so \( p \) would be justified in \( P' \).

Let us now consider **RED**\( _{neg}^- \). This transformation rule holds for both defeasible and strict rules in a **DEL**\( _{neg} \) program \( P \), as shown in Proposition 4.2.

**Proposition 4.2.** Let \( P \) be a **DEL**\( _{neg} \) program. Let \( P' \) be the resulting program of applying **RED**\( _{neg}^- \), i.e. \( P \leftarrow_{N neg} P' \). Then \( \text{SEM}_{DelP}(P') = \text{SEM}_{DelP}(P) \).

**Proof.** Let \( P \) be a **DEL**\( _{neg} \) program, and let \( A \leftarrow \in P \). Let \( rP \leftarrow Q_1, \ldots, Q_n \) (resp. \( P \leftarrow Q_1, \ldots, Q_n \)) be a rule in \( P \), such that \( \sim A \equiv Q_i \), for some \( i \). Then \( r \) cannot be used in any defeasible derivation corresponding to an argument in \( P \), since if \( r \) is used, then both \( A \) and \( A \) follow from \( \Pi \cup A \), contradicting the definition of argument). Then, every argument that can be built from \( P \) can also be built from \( P' = P - \{ r \} \). Thus \( \text{Arg}(P) = \text{Arg}(P') \), and therefore \( \text{SEM}_{DelP}(P') = \text{SEM}_{DelP}(P) \).

Let us now consider **SUB**\( _{neg} \). This transformation holds for strict rules, as shown in Proposition 4.4. It does not hold in **DEL**\( _{neg} \) for defeasible rules (since having more literals in the body gives more specific information), as shown in Example 4.3

**Example 4.3.** Let \( P = (\Pi, \Delta) \), where \( \Pi = \{ q_1, q_2 \} \) and \( \Delta = \{ (p \leftarrow < q_1, q_2), (p \leftarrow q_1), (\sim p \leftarrow < q_2) \} \). The argument \( A = \{ (p \leftarrow < q_1, q_2) \} \) for \( p \) is strictly more specific than \( B = \{ (\sim p \leftarrow < q_2) \} \) for \( \sim p \). However, if we consider \( P' = P - \{ (p \leftarrow < q_1, q_2) \} \), then we get two arguments with block each other (\( A = \{ (p \leftarrow < q_1) \} \) for \( p \) and \( B = \{ (\sim p \leftarrow < q_2) \} \) for \( \sim p \)).

**Proposition 4.4.** Let \( P \) be a **DEL**\( _{neg} \) program. Let \( P' \) be the program resulting from applying **SUB**\( _{neg} \), i.e. \( P \leftarrow_{M neg} P' \). Then \( \text{SEM}_{DelP}(P) = \text{SEM}_{DelP}(P') \).

**Proof.** Clearly, \( P' = P - \{ r | r \) is a non-minimal rule \}. Let \( rP \leftarrow Q_1, \ldots, Q_k \) be a non-minimal rule in \( P \), and assume there is an argument \( A \) for some literal \( H \) in which \( r \) is part of the defeasible derivation for \( H \). From the definition of defeasible derivation, for every literal \( Q_1, \ldots, Q_k \) there is a rule \( (B_1, Q_1) \) \( \ldots \) \( (B_k, Q_k) \), such that \( \bigcup_{i=1}^{k} B_i \subseteq A \). Since \( r \) is a non-minimal rule, there exists \( r' = P \leftarrow Q_j, \ldots, Q_{i} \in \Pi, j < k \), such that for every literal \( Q_i \) (\( i = 1 \ldots j \)) there are arguments \( (B_1, Q_1) \) \( \ldots \) \( (B_j, Q_j) \). But \( \bigcup_{i=1}^{j} B_i \subseteq \bigcup_{i=1}^{k} B_k \). Hence by replacing \( r \) by \( r' \) we get either the same set \( A \) as an argument for \( H \), or a proper subset \( A' \subseteq A \) as an argument for \( H \). In any case, the rule \( r \) can be removed from \( P \), without affecting the arguments that can be obtained from \( P \). Therefore \( \text{Arg}(P) = \text{Arg}(P') \), with \( P' = P - \{ r \} \). Hence \( \text{SEM}_{DelP}(P) = \text{SEM}_{DelP}(P') \).
Let us now consider UNFOLD$_{neg}$. This property does not hold for defeasible rules, as shown in Example 4.5. Besides, it does not hold for strict rules in general either: we impose the additional condition that no defeasible rule has the same head as the literal which is being removed by applying UNFOLD$_{neg}$. The reason for doing so is shown in Example 4.6.

**Example 4.5 (UNFOLD Does not Hold for Defeasible Rules).** Consider the following example

\[ \Pi \quad \Delta \]
\[
\begin{align*}
\text{has feathers} & \leftarrow \text{flies} \leftarrow \text{bird} \\
\text{has beak} & \leftarrow \sim \text{flies} \leftarrow \text{bird}, \text{wounded} \\
\text{wounded} & \leftarrow \text{bird} \leftarrow \text{has feathers}, \text{has beak}
\end{align*}
\]

In \( \mathcal{P} \), there is an argument \( A_1 = \{ \sim \text{flies} \leftarrow \text{bird}, \text{wounded} \} \) for \( \sim \text{flies} \) which is strictly more specific than \( A_2 = \{ \text{flies} \leftarrow \text{bird}, (\sim \text{flies} \leftarrow \text{bird}, \text{wounded}) \} \) for \( \text{flies} \). In this case, the first argument is a justification. However, if UNFOLD$_{neg}$ is applied on defeasible rules, we get \( \mathcal{P}' = (\Pi, \Delta') \), with \( \Delta' = \{ \text{flies} \leftarrow \text{has feathers}, \text{has beak}, \sim \text{flies} \leftarrow \text{bird}, \text{wounded}, \text{bird} \leftarrow \text{has feathers}, \text{has beak} \} \). In \( \mathcal{P}' \) we have two conflicting arguments, \( A_1 = \{ (\sim \text{flies} \leftarrow \text{bird}, \text{wounded}), (\text{bird} \leftarrow \text{has feathers}, \text{has beak}) \} \) for \( \sim \text{flies} \) and \( A_2 = \{ \text{flies} \leftarrow \text{has feathers}, \text{has beak} \} \) for \( \text{flies} \). In this case, neither of them is strictly more specific than the other.

**Example 4.6.** Let \( \mathcal{P} = (\Pi, \Delta) \) be a dlp, where \( \Pi = \{ (p \leftarrow q, s), (q \leftarrow f_1), (g \leftarrow f_2), (s \leftarrow t), (t \leftarrow s) \} \), and \( \Delta = \{ q \leftarrow s \} \). If we could apply UNFOLD$_{neg}$ on rule \( p \leftarrow q, s \), we would get the program \( \mathcal{P}' = \mathcal{P} - \{ p \leftarrow q, s \} \cup \{ p \leftarrow f_1, s, p \leftarrow f_2, s \} \). But \( A_1 = \{ q \leftarrow t \} \) is an argument for \( p \) in \( \mathcal{P} \), but it does not exist in \( \mathcal{P}' \).

**Proposition 4.7.** Let \( \mathcal{P} = (\Pi_G \cup \Pi_C, \Delta) \) be a dlp, such that \( \Pi = \Pi_G \cup \Pi_C \). \( \Pi_C \) denotes the strict rules (excluding facts) in \( \Pi \), and \( \Pi_G \) the set of facts in \( \Pi \). Let \( \mathcal{P}^+ \) denote all possible literals that have a defeasible derivation from \( \mathcal{P} \). Then for any \( F \subseteq \Pi_C \), \( (\Pi_G \cup F)^+ = (\Pi_G \cup F)^+ \), where \( \Pi_G \) follows from \( \Pi_G \) by UNFOLD$_{neg}$ (i.e., \( \Pi_G \rightarrow \Pi_G \)).

**Proof.** (Sketch) Since we do not consider defeasible rules, all defeasible derivations are actually 'strict' (i.e. not involving defeasible information). Assume \( \Pi_G \neq \Pi_G^+ \), where \( \Pi_G \rightarrow \Pi_G^+ \) (otherwise our conclusion trivially holds). Then there exists at least a pair of strict rules \( r_i, r_{i+1} \) which are UNFOLD$_{neg}$-related. We will show that any defeasible derivation \( S \) in \( (\Pi_G \cup F) \) involving such rules has its counterpart in \( \Pi_G^+ \cup F \), and vice versa. Let \( S = [r_1, r_2, \ldots, r_i, r_{i+1}, \ldots, r_n] \) be the sequence of rules used for deriving \( H \in (\Pi_G \cup F) \). Consider the subsequence \( [r_i, r_{i+1}] \), such that \( r_i \) and \( r_{i+1} \) are UNFOLD$_{neg}$-related. Let \( r_i = A \leftarrow B \), and let \( r_{i+1} = C \leftarrow B' \). Let UNFOLD$_{neg}(P)$ be the program resulting from applying UNFOLD$_{neg}$ to a program \( \mathcal{P} \). Clearly, the subsequence \( [r_i, r_{i+1}] \) is no longer valid in UNFOLD$_{neg}(\Pi_G \cup F)$, since \( r_i \) was removed. However, by applying UNFOLD$_{neg}$ we have substituted it by \( [r_i^k] \), where \( r_i^k \) is the instance obtained by replacing \( r_i \) by \( A \leftarrow (B - \{C\} \cup B') \). Clearly, \( r_i^k \) has the same subgoals as \( [r_i, r_{i+1}] \), and its head coincides with the head of \( r_i \). It follows that \( S' = [r_1, r_2, \ldots, r_i^k, \ldots, r_n] \) is a defeasible derivation for \( h \) in UNFOLD$_{neg}(\Pi_G \cup F)$.

**Proposition 4.8.** Let \( \mathcal{P} \) be a dlp, and let \( \mathcal{P} \rightarrow \Pi_G \mathcal{P}' \). Then \( \langle A, H \rangle \in \text{Args}(\mathcal{P}) \iff \langle A, H \rangle \in \text{Args}(\mathcal{P}') \)

---

8 For space reasons we do not include the proof in detail; the interested reader is referred to [CDSS00].
Proof. Assume $\langle A, H \rangle$ is an argument in a dlp $P = (\Pi, \Delta)$. Then $\Pi \cup A \vdash H$, or equivalently $\Pi_G \cup \Pi_C \cup A \vdash H$. But from Proposition 4.7 this is equivalent to $\Pi' \cup A \vdash H$, where $\Pi_G \cup \Pi_C \vdash_{un}$ $\Pi'$. Clearly, this defeasible derivation is non-contradictory, and minimal. Hence $\langle A, H \rangle \in \text{Args}(P')$

Corollary 4.9. Let $P'$ be the program resulting of applying $\text{UNFOLD}_{\neg}$, i.e. $P \vdash_{\neg} P'$. Then $\text{SEM}_{\text{DeLP}}(P) = \text{SEM}_{\text{DeLP}}(P')$.

Let us now consider tautology elimination.

Proposition 4.10. Let $P$ be a $\text{DeLP}_{\neg}$ program, and $P'$ the program resulting from applying $\text{TAUT}_{\neg}$ to $P$, i.e. $P \vdash_{\neg} P'$ Then $\text{SEM}_{\text{DeLP}}(P) = \text{SEM}_{\text{DeLP}}(P')$.

Proof. Let $\langle A, Q \rangle$ be an argument in $\text{Args}(P)$, such that $\Pi \cup A \vdash Q$ using a strict rule $r = P \leftarrow Q_1, \ldots, Q_k$. Then the occurrence of $P$ in the antecedent can also be proven from $\Pi - \{r\} \cup A$. Thus, there exists a derivation for $Q$ from $\Pi - \{r\} \cup A$ (the same holds the other way around). Therefore, $\langle A, Q \rangle \in \text{Args}(P)$ iff $\langle A, Q \rangle \in \text{Args}(P - \{r\})$. Assume now that $\langle A, P \rangle$ is an argument in $\text{Args}(P)$, such that $\Pi \cup A \vdash P$ using a defeasible rule $r = P \rightarrow P, S_1, \ldots, S_k$. Let $A' = A \setminus \{r\}$. Clearly, $\Pi \cup A' \vdash P$. But then $\langle A, P \rangle$ is not an argument, since it is not minimal (contradiction). Therefore, no defeasible rule $P \rightarrow P, S_1, \ldots, S_k$ can be used in building an argument. Therefore, $\langle A, P \rangle \in \text{Args}(P)$ iff $\langle A, P \rangle \in \text{Args}(P - \{r\})$.

4.2 Transformation Properties in $\text{DeLP}_{\neg}$

$\text{DeLP}_{\neg}$ can be seen as $\text{NLP}$ with the addition of defeasible rules. In such a setting there is no strict negation “$\neg$”, and therefore no contradictory literals $P$ and $\neg P$. The attack relationship among arguments is completely captured by the semantics of default negation in $\text{DeLP}$: $\neg H$ holds iff $H$ cannot be justified [GSC98]. In this respect, $\text{DeLP}$ naturally extends the intended meaning of default negation in traditional logic programming ($\neg H$ holds iff $H$ fails to be finitely proven).\(^9\)

Since a $\text{DeLP}_{\neg}$ program does not involve strict negation, many problems considered in Subsection 4.1 do not arise. New transformations $\text{RED}^+_{\neg}$, $\text{RED}^-_{\neg}$, $\text{SUB}_{\neg}$, $\text{UNFOLD}_{\neg}$ and $\text{TAUT}_{\neg}$ can be defined, with the same meaning as the ones introduced in Subsection 4.1, but referring to default negation. A complete analysis of transformations for $\text{DeLP}_{\neg}$ is outside the scope of this paper (the interested reader is referred to [CDSS00]). For every transformation, we will show that the resulting transformed program is equivalent to the original one.

1. (RED\(^+\)_\neg): Let $\langle A, H \rangle$ be an argument in $P$, such that $r = P \leftarrow Q_1, \ldots, Q_k$ is a strict rule used in the defeasible derivation of $H$ in $A$. The literal $\neg Q, \ldots, Q_k$ is not justified. Since there is no rule with head $Q$ in $P$, there is no argument for $Q$, and hence no justification for $Q$. Therefore $\langle A, H \rangle$ is also an argument in $P - \{r\}$, with $r' = P \leftarrow Q_1, \ldots, Q_k$. The same applies for any other strict or defeasible rule used in the defeasible derivation. Hence if $P \vdash_{\neg} P'$, then $\text{SEM}_{\text{DeLP}}(P') = \text{SEM}_{\text{DeLP}}(P')$.

2. (RED\(^-\)_\neg): Let $r = P \leftarrow Q_1, \ldots, Q_n$ be a strict rule, and assume $Q \leftarrow P$. If $r$ is used in a defeasible derivation for building an argument $\langle A, H \rangle$, the literal $\neg Q$ will hold iff $Q$ is not justified. But the empty argument $\langle \emptyset, Q \rangle$ is a justification for $Q$. Hence $r$ cannot

\(^9\) Note that default negation is applied to positive literals (i.e., atoms) in $\text{NLP}$, whereas in $\text{DeLP}$ it can be applied to arbitrary literals.
<table>
<thead>
<tr>
<th>NLP under wfs</th>
<th>DeLP_{neg}</th>
<th>DeLP_{not}</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED^{+}</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>RED^{-}</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SUB</td>
<td>yes</td>
<td>yes, for strict rules</td>
</tr>
<tr>
<td>UNFOLD</td>
<td>yes</td>
<td>yes^{a}, for strict rules</td>
</tr>
<tr>
<td>TAUT</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Fig. 1. Behavior of NLP, DeLP_{neg} and DeLP_{not} under different transformations

^{a} Some additional conditions are required for the transformation to hold.

be used in any argument, so that any argument \( \langle A, H \rangle \) in \( \text{Args}(P) \) is also an argument in \( \text{Args}(P \setminus \{r\}) \). The same applies for any other strict or defeasible rule used in the defeasible derivation of \( H \) in \( A \). Therefore if \( P \rightarrow_{P_{\neg}} P' \), then \( \text{SEM}_{\text{DeLP}}(P) = \text{SEM}_{\text{DeLP}}(P') \).

3. (SUB\textsubscript{not}): Let \( P \) be a DeLP\textsubscript{not} program, and let \( r = P \leftarrow B_1 \) be a non-minimal rule in \( P \) (i.e., there exists a rule \( r' = P \leftarrow B_2 \) such that \( B_2 \subseteq B_1 \)). If there is an argument \( A \) for \( H \) using rule \( r \) in the defeasible derivation of \( H \), then the same argument \( A \) for \( H \) can be built by using \( r' \) instead (since every literal \( \text{not} \ Q \) that holds in \( B_2 \) also holds in \( B_1 \)). Hence \( \text{Args}(P) = \text{Args}(P \setminus \{r\}) \). The same applies for any other strict rule used in the defeasible derivation of \( h \) in \( A \). Therefore SUB\textsubscript{not} holds wrt strict rules.

4. (UNFOLD\textsubscript{not}): this transformation holds in DeLP\textsubscript{not}, following the same line of reasoning used in Proposition 4.8.

5. (TAUT\textsubscript{not}): tautology elimination holds in DeLP\textsubscript{not}, following the same line of reasoning used in Proposition 4.10.

5 Relating NLP and DeLP

Figure 1 summarizes the behavior of NLP, DeLP\textsubscript{neg} and DeLP\textsubscript{not} under the different transformation rules presented before. From that table we can identify some relevant features:

- An argumentation-based semantics has been given to NLP using an abstract argumentation framework [KT99]. From Section 4.2 it is clear that DeLP is a proper extension of NLP, since there are transformation properties in NLP which do not hold in DeLP. This is basically due to the knowledge representation capabilities provided by defeasible rules.

- Some properties of NLP under well-founded semantics are also present in DeLP (such as TAUT and RED^{-}). It is worth noticing that RED^{-} holds in NLP because of a ‘consistency constraint’ (it cannot be the case that both not P and P hold). The same is achieved in DeLP by demanding non-contradiction when constructing arguments.

- Other transformation properties only hold for strict rules (e.g. SUB), sometimes with extra requirements (e.g. UNFOLD). This shows that defeasible rules express a link between literals that cannot be easily ‘simplified’ in terms of a transformation rule, and a more complex
analysis (e.g. computing defeat) is required.

- Some properties (e.g. RED\(^+\)) do not hold at all wrt strict negation, but do hold wrt default negation. In the first case, the reason is that negated literals are treated as new predicate names (and succeed as subgoals iff they can be proven from the program). In the second case, default negation behaves much like its counterpart in NLP. As in NLP, the absence of rules with head \(H\) is enough for concluding that \(H\) cannot be proven, and therefore not justified.

5.1 Related work

In recent work [KT99] an abstract argumentation framework has been used as a basis for defining an unifying proof theory for various argumentation semantics of logic programming. In that framework, well-founded semantics for NLP is computed by using an argument-based approach, which has many similarities with DeLP [CS99].

Many semantics for extended logic programs view default negation and symmetric negation as unrelated. To overcome this situation a semantics WFSX for extended logic programs was defined [ADP95]. Well-founded Semantics with Explicit Negation (WFSX) embeds a “coherence principle” providing the natural missing link between both negations: if \(\sim L\) holds then \(\text{not } L\) should hold too (similarly, if \(L\) then \(\text{not } \sim L\)). In DeLP this “coherence principle” also holds [GSC98].

Finally, it must be remarked the the original Simari-Loui formulation [SL92] contains a fixed-point definition that characterizes all justified beliefs. A similar approach was used later by Prakken & Sartor [PS97] in an extended logic programming setting, getting a revised version of well-founded semantics as defined by Dung [Dun93]. These analogies highlight the link between well-founded semantics and skeptical argumentative frameworks.

6 Conclusion

We have related in this paper the logical framework DeLP to classical logic programming semantics, particularly well-founded semantics for NLP. The link between both semantics was established by looking for analogies and differences in the results of applying transformation rules on logic programs.

The differences between NLP and DeLP are to be found in the expressive power of DeLP for encoding knowledge in comparison with NLP. Defeasible rules allow the formalization of criteria for defeat among arguments which cannot be easily ‘compressed’ by applying transformation rules, as explained in Section 5. Strict negation in DeLP is also a feature which extends the representation capabilities of NLP. However, as already discussed, the same principle which guides the application of the transformation rule RED\(^-\) in NLP can be used for detecting rules that cannot be used for constructing arguments.

It is worth noting that the original motivation for DeLP was to find an argumentative formulation for defeasible theories in order to resolve potential inconsistencies. This was at the end of the 80’s. In the meantime the area of semantics for logic programs underwent a solid foundational phase and today several possible semantics together with their properties are well-known. We contend that these results can be applied to gain a better understanding of argumentation-based frameworks.
References


