Relating Defeasible and Normal Logic Programming through Transformation Properties[⋆]

Carlos Chesñevar, cic@cs.uns.edu.ar[1]
Jürgen Dix, dix@uni-koblenz.de[2]
Frieder Stolzenburg, stolzen@uni-koblenz.de[2]
Guillermo Simari, grs@cs.uns.edu.ar[1]

[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. The semantics SEM_{DeLP} is defined using the notion of a *dialectical tree*. In contrast to this, in *normal logic programming* (*NLP*), there is one negation *not*, interpreted as a kind of *negation-as-failure* and all rules can be considered defeasible. In *ENLP*, the class *NLP* is extended by another form of negation, which is usually considered as *explicit* negation denoted by "$\sim$". Various semantics have been defined for *NLP* (resp. *ENLP*), notably the well-founded semantics WFS (resp. WFSX).
In particular, we consider the *transformation properties* 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*.

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.[1] 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 $not\, B$. If $\mathcal{B} = \{L_1, \ldots, L_n\}$ is the body of a rule $A \leftarrow \mathcal{B}$, we also use the notation $A \leftarrow \mathcal{B}^+, not\, \mathcal{B}^-$, where $\mathcal{B}^+$ (resp. $\mathcal{B}^-$) contains all the positive (resp. negative) body atoms in $\mathcal{B}$.*

In *NLP*, atoms A and negated atoms $not\, A$ 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*, refering to an atom A and its default negation $not\, A$. If $\mathcal{B}^+ = \mathcal{B}^- = \emptyset$, 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 $\mathrm{Prog}_{\mathcal{L}}$ 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 $I_1 \cup not\, I_2$.

Definition 2.2 (Semantics SEM). *A semantics SEM is a mapping which assigns to every logic program $\mathcal{P}$ a set SEM ($\mathcal{P}$) of (partial) models of $\mathcal{P}$, such that SEM is "instantiation invariant", i.e. SEM($\mathcal{P}$) = SEM($ground(\mathcal{P})$), where $ground(\mathcal{P})$ denotes the Herbrand instantiation of $\mathcal{P}$. A semantics SEM is called* 3-value based *if for every program $\mathcal{P}$ the partial interpretation 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 $\sim p$, where the symbol "$\sim$" represents *strong negation*. We define this formally:

Definition 2.3 (Strict, $\leftarrow$, and Defeasible Rules, $-\!\!\prec$). *A* strict rule *(*defeasible rule*) is an ordered pair, conveniently denoted by Head $\leftarrow$ Body (Head $-\!\!\prec$ Body), whose first member, Head, is a literal, and whose second member, 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 $L_0 \leftarrow L_1, \ldots, L_n$ ($L_0 -\!\!\prec L_1, \ldots, L_n$). If the body is empty, it is written $L \leftarrow true$ ($L -\!\!\prec true$), and it is called a* fact *(*presumption*). Facts may also be written as L.*

[2] Following [Lif94], program rules with variables are viewed as "schemata" that represent their ground instances.
[3] We equip $\leftarrow$ with the Kleene interpretation, where undef $\leftarrow$ undef is considered to be true.

In the sequel, atoms will be denoted with lowercase letters (a, b, ...). 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, ...), possibly subindicated. Sets will be denoted as $\mathcal{A}$, $\mathcal{B}$, ... , possibly subindicated. Logic programs will be usually denoted as $\mathcal{P}_1$, $\mathcal{P}_2$, 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 Π of strict rules, and the subset Δ of defeasible rules. When required, we will denote $\mathcal{P}$ as (Π, Δ). We will distinguish the class of all defeasible logic programs that use only strict (resp. default) negation, denoting them as DeLP$_{neg}$ (DeLP$_{not}$, resp.).*

From Dix to all:

(What does it mean that a literal **follows**?) A set of rules $\mathcal{S}$ is *contradictory* iff a pair of contradictory literals P and $\sim P$ follows from $\mathcal{S}$. Given a *dlp* $\mathcal{P}$, we will assume that the set Π of strict rules is non-contradictory.[4] Given a *dlp* $\mathcal{P}$, 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* $\mathcal{P}$. The symbol "$\sim$" is considered as part of the predicate when generating a defeasible derivation.

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 $L = L\sigma$ for some ground variable substitution σ, 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 $\mathcal{P}$, 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 $Args(\mathcal{P})$ 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 an 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*. To establish if $\mathcal{A}$ is a non-defeated argument, counterarguments that could be *defeaters* for $\mathcal{A}$ are considered, i. e. counterarguments that for some criterion are preferred to $\mathcal{A}$. *DeLP* considers a particular 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 $\langle \mathcal{A}_1, Q_1 \rangle$ defeats $\langle \mathcal{A}_2, Q_2 \rangle$ at a literal Q iff there exists a subargument $\langle \mathcal{A}, Q \rangle$ of $\langle \mathcal{A}_2, Q_2 \rangle$ such that $\langle \mathcal{A}_1, Q_1 \rangle$ counterargues $\langle \mathcal{A}_2, Q_2 \rangle$ at Q, and either: (a) $\langle \mathcal{A}_1, Q_1 \rangle$ is "better" that $\langle \mathcal{A}, Q \rangle$ (then $\langle \mathcal{A}_1, Q_1 \rangle$ is a proper defeater of $\langle \mathcal{A}, Q \rangle$); or (b) $\langle \mathcal{A}_1, Q_1 \rangle$ is unrelated by the preference order to $\langle \mathcal{A}, Q \rangle$ (then $\langle \mathcal{A}_1, Q_1 \rangle$ is a blocking defeater of $\langle \mathcal{A}, Q \rangle$).*

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

1. *A single node labeled with an argument $\langle \mathcal{A}, Q \rangle$ with no defeaters (proper or blocking) is by itself the dialectical tree for $\langle \mathcal{A}, Q \rangle$.*
2. *Let $\langle \mathcal{A}_1, Q_1 \rangle, \langle \mathcal{A}_2, Q_2 \rangle, \ldots, \langle \mathcal{A}_n, Q_n \rangle$ be all the defeaters (proper or blocking) for $\langle \mathcal{A}, Q \rangle$. We construct the dialectical tree for $\langle \mathcal{A}, Q \rangle$, $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$, by labeling the root node with $\langle \mathcal{A}, Q \rangle$ and by making this node the parent node of the roots of the dialectical trees for $\langle \mathcal{A}_1, Q_1 \rangle, \langle \mathcal{A}_2, Q_2 \rangle, \ldots, \langle \mathcal{A}_n, Q_n \rangle$.*

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

1. *All the leaves in $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$ are marked as U-nodes.*
2. *Let $\langle \mathcal{B}, H \rangle$ be an inner node of $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$. Then $\langle \mathcal{B}, H \rangle$ will be a U-node iff every child of $\langle \mathcal{B}, H \rangle$ is a D-node. The node $\langle \mathcal{B}, H \rangle$ 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.*[6] An argument $\mathcal{A}$ which turns to be ultimately undefeated is called a *justification*. Formally:

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

A given query Q can be associated with a particular *answer set* according to some criterion. Several criteria have been analyized 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 $\mathcal{P}$, 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 $\langle \mathcal{A}, Q \rangle$.*
2. *Q is a negative answer iff for every argument $\langle \mathcal{A}, Q \rangle$, in the dialectical tree $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$, there exists at least a proper defeater for $\mathcal{A}$ marked as U.*
3. *Q is an undecided answer iff Q is not justified, and for every argument $\langle \mathcal{A}, Q \rangle$, it is the case that $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$ has at least one blocking defeater marked as U.*
4. *Q is an unknown answer if there is no argument for Q.*

Given a dlp $\mathcal{P}$, we call $\mathsf{Positive}(\mathcal{P})$, $\mathsf{Negative}(\mathcal{P})$, $\mathsf{Undefined}(\mathcal{P})$ *and* $\mathsf{Unknown}(\mathcal{P})$ *the sets of positive, negative, undecided and unknown answers, resp.*

[6] For space reasons we do not discuss these conditions in this paper; see [GSC98] for an in-depth analysis.

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

Definition 2.13 (SEM_{DeLP}).
For any dlp $\mathcal{P}$, we define SEM_{DeLP} $(\mathcal{P})$ = $\langle \mathcal{P}^{accepted}, \mathcal{P}^{rejected}, \mathcal{P}^{undef}\rangle$, where $\mathcal{P}^{accepted}$ = $\{Q|Q \in \mathsf{Justified}(\mathcal{P})\}$, $\mathcal{P}^{rejected}$ = $\{Q|Q \in \mathsf{Unknown}(\mathcal{P}) \cup \mathsf{Negative}(\mathcal{P})\}$, and $\mathcal{P}^{undef}$ = $\{Q|Q \in \mathsf{Undefined}(\mathcal{P})\}$.

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 $Args(\mathcal{P}) = Args(\mathcal{P}')$. This equivalence will be frequently used in the following sections.

3 Transformations for *NLP*: classifying well-founded semantics

A *program transformation* is a relation $\mapsto$ between ground logic programs [BDFZ01]. A semantics SEM allows a transformation $\mapsto$ iff $\text{SEM}(\mathcal{P}_1) = \text{SEM}(\mathcal{P}_2)$, for all $\mathcal{P}_1$ and $\mathcal{P}_2$, such that $\mathcal{P}_1 \mapsto \mathcal{P}_2$. In this case we also say that the transformation $\mapsto$ *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 $\mathcal{P} \in Prog_{\mathcal{L}}$, let $HEAD(\mathcal{P})$ be the set of all head-atoms of $\mathcal{P}$, i.e. $HEAD(\mathcal{P}) = \{H | H \leftarrow \mathcal{B}^+, not\,\mathcal{B}^- \in \mathcal{P}\}$. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be ground programs. The following transformation rules characterize WFS:*

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

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

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

UNFOLD: *(Unfolding) Program $\mathcal{P}_2$ results from program $\mathcal{P}_1$ by **UNFOLD** (written $\mathcal{P}_1 \mapsto_U \mathcal{P}_2$) iff there is a rule $H \leftarrow \mathcal{B}$ in $\mathcal{P}_1$ and a positive literal $B \in \mathcal{B}$ such that $\mathcal{P}_2 = \mathcal{P}_1$ - $\{H \leftarrow \mathcal{B}\} \cup \{H \leftarrow ((\mathcal{B} - \{B\}) \cup \mathcal{B}') \mid B \leftarrow \mathcal{B}' \in \mathcal{P}_1\}$*

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

A program $\mathcal{P}'$ is a normal form *of a program $\mathcal{P}$ wrt a transformation "$\mapsto$" iff $\mathcal{P} \mapsto_* \mathcal{P}'$, and $\mathcal{P}'$ is irreducible, i.e. there is no program $\mathcal{P}''$ such that $\mathcal{P}' \mapsto \mathcal{P}''$.*

Let "$\mapsto_R$" be the rewriting system consisting of the above five transformations, i.e. $\mapsto_R = \mapsto_T \cup \mapsto_U \cup \mapsto_M \cup \mapsto_P \cup \mapsto_N$. 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 $norm_{\text{WFS}}(\mathcal{P})$). This normal form $norm_{\text{WFS}}(\mathcal{P})$ 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 ($\mathbf{SEM_{min}}$). *For any nlp $\mathcal{P}$, we define $SEM_{min}(\mathcal{P}) = \langle \mathcal{P}^{true}, \mathcal{P}^{false}, \mathcal{P}^{undef} \rangle$, where $\mathcal{P}^{true} = \{H | H \leftarrow \in \mathcal{P}\}$, $\mathcal{P}^{false} = \{H | H \in \mathcal{L}_P - HEAD(\mathcal{P})\}$, and $\mathcal{P}^{undef} = \{H | H \in \mathcal{L}_P - (\mathcal{P}^{true} \cup \mathcal{P}^{false})\}$,*

Theorem 3.3 (Classifying WFS [BD99]). *$WFS(\mathcal{P}) = SEM_{min}(norm_{WFS}(\mathcal{P}))$.*

4 Transformation Properties in *DeLP*

As stated in the introduction, we want to analyze whether transformations as the ones described above also hold for a *DeLP* program. In our analysis, we will focus first on $DeLP_{neg}$ (i.e., *DeLP* with strict negation "$\sim$"). As the transformations in [BDFZ01] are defined wrt a *NLP* setting, we will adapt them accordingly. Therefore, we extend our previous terminology to be applied to a $DeLP_{neg}$ program $\mathcal{P}$ (thus HEAD $(\mathcal{P})$ stands for all heads of rules in $\mathcal{P}$, etc.), distinguishing strict rules from defeasible rules when needed. Next, we will consider $DeLP_{not}$ (*DeLP* with default negation *not*). In this case, a similar analysis will be performed.

4.1 Transformation Properties in $\boldsymbol{DeLP_{neg}}$

Below we will introduce tentative extensions to $DeLP_{neg}$ 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.

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

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

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

$\mathbf{UNFOLD}_{neg}$: Suppose program $\mathcal{P}_1$ contains a <u>strict</u> rule $H \leftarrow \mathcal{B}$ such that <u>there is no defeasible rule in $\mathcal{P}_1$ with head H</u>. Then program $\mathcal{P}_2$ will result from program $\mathcal{P}_1$ by $\mathbf{UNFOLD}_{neg}$ (written $\mathcal{P}_1 \mapsto_{Uneg} \mathcal{P}_2$) iff there is a positive literal $B \in \mathcal{B}$[7] such that $\mathcal{P}_2 = \mathcal{P}_1$ - $\{H \leftarrow \mathcal{B}\} \cup \{H \leftarrow ((\mathcal{B} - \{B\}) \cup \mathcal{B}') \mid B \leftarrow \mathcal{B}' \in \mathcal{P}_1\}$. The clause $H \leftarrow \mathcal{B}$ is said to be ***UNFOLD**$_{neg}$-related* with each $B \leftarrow \mathcal{B}_i \in \mathcal{P}_1$ (for $i = 1, \ldots, n$).

$\mathbf{TAUT}_{neg}$: Program $\mathcal{P}_2$ will result from program $\mathcal{P}_1$ by $\mathbf{TAUT}_{neg}$ (written $\mathcal{P}_1 \mapsto_{Tneg} \mathcal{P}_2$) iff there is $H \leftarrow \mathcal{B} \in \mathcal{P}_1$ such that $H \in \mathcal{B}$ and $\mathcal{P}_2 = \mathcal{P}_1 - \{H \leftarrow \mathcal{B}\}$.

First we consider $\mathbf{RED}^+_{neg}$. This transformation rule *does not hold* for strict negation. Note that whereas $\mathbf{RED}^+$ 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:

[7] Note that we do not distinguish between atoms and their negations because negated literals are treated as new predicate names.

Example 4.1. Consider the following $DeLP_{neg}$ program: $\Pi = \{ (p \leftarrow \sim s), (\sim s \leftarrow t), (q_1 \leftarrow), (q_2 \leftarrow) \}$ and $\Delta = \{ (t \prec q_1), (\sim t \prec q_1, q_2) \}$. Here p is not justified from $\mathcal{P}$ (since the argument $\mathcal{A}_1 = \{ t \prec q_1 \}$ for p is defeated by the argument $\mathcal{A}_2 = \{ \sim t \prec q_1, q_2 \}$ for $\sim t$. If we consider $\mathcal{P}' = \mathbf{RED}^+_{neg}(\mathcal{P})$ we get p as a fact, so p would be justified in $\mathcal{P}'$.

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

Proposition 4.2. *Let $\mathcal{P}$ be a $DeLP_{neg}$ program. Let $\mathcal{P}'$ be the resulting program of applying $\boldsymbol{RED}^-_{neg}$, i.e. $\mathcal{P} \mapsto_{Nneg} \mathcal{P}'$. Then $SEM_{\mathrm{DeLP}}(\mathcal{P}') = SEM_{\mathrm{DeLP}}(\mathcal{P})$.*

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

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

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

Proposition 4.4. *Let $\mathcal{P}$ be a $DeLP_{neg}$ program. Let $\mathcal{P}'$ be the program resulting from applying $\boldsymbol{SUB}_{neg}$, i.e. $\mathcal{P} \mapsto_{Mneg} \mathcal{P}'$. Then $SEM_{\mathrm{DeLP}}(\mathcal{P}) = SEM_{\mathrm{DeLP}}(\mathcal{P}')$.*

Proof. Clearly, $\mathcal{P}' = \mathcal{P}$ - $\{ r \mid r$ is a non-minimal rule $\}$. Let $r = P \leftarrow Q_1 \ldots Q_k$ be a non-minimal rule in $\mathcal{P}$, and assume there is an argument $\mathcal{A}$ 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 an argument $\langle \mathcal{B}_1, Q_1 \rangle \ldots \langle \mathcal{B}_k, Q_k \rangle$, such that $\bigcup_{i=1}^{k} \mathcal{B}_i \subseteq \mathcal{A}$. Since r is a non-minimal rule, there exists $r' = P \leftarrow Q_1 \ldots Q_j \in \Pi$, $j < k$, such that for every literal Q_i $(i = 1..j)$ there are arguments $\langle \mathcal{B}_1, Q_1 \rangle \ldots \langle \mathcal{B}_j, Q_j \rangle$. But $\bigcup_{i=1}^{j} \mathcal{B}_i \subseteq \bigcup_{i=1}^{k} \mathcal{B}_k$. Hence by replacing r by r' we get either the same set $\mathcal{A}$ as an argument for H, or a proper subset $\mathcal{A}' \subset \mathcal{A}$ as an argument for H. In any case, r can be removed from $\mathcal{P}$, without affecting the arguments that can be obtained from $\mathcal{P}$. Therefore $Args(\mathcal{P}) = Args(\mathcal{P}')$, with $\mathcal{P}' = \mathcal{P} - \{r\}$. Hence $\mathrm{SEM}_{DeLP}(\mathcal{P}) = \mathrm{SEM}_{DeLP}(\mathcal{P}')$.

Let us now consider $\mathbf{UNFOLD}_{neg}$. 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 $\mathbf{UNFOLD}_{neg}$. 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

Π	Δ
$has\_feathers \leftarrow$	$flies \prec bird$
$has\_beak \leftarrow$	$\sim flies \prec bird, wounded$
$wounded \leftarrow$	$bird \prec has\_feathers, has\_beak$

In $\mathcal{P}$, there is an argument $\mathcal{A}_1 = \{ (\sim flies \prec bird, wounded), (bird \prec has\_feathers, has\_beak)\}$ for $\sim flies$ which is strictly more specific than $\mathcal{A}_2 = \{ (flies \prec bird), (bird \prec has\_feathers, has\_beak) \}$ for $flies$. 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 Δ'=$\{ (flies \prec has\_feathers, has\_beak), (\sim flies \prec bird, wounded), (bird \prec has\_feathers, has\_beak)\}$. In $\mathcal{P}'$ we have two conflicting arguments, $\mathcal{A}_1 = \{ (\sim flies \prec bird, wounded), (bird \prec has\_feathers, has\_beak)\}$ for $\sim flies$ and $\mathcal{A}_2 = \{ (flies \prec has\_feathers, has\_beak) \}$ for $flies$. 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 $\Pi = \{ (p \leftarrow q, s), (q \leftarrow f_1), (q \leftarrow f_2), (s \leftarrow), (t \leftarrow) \}$, and $\Delta = \{ q \prec s \}$. 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 $\mathcal{A}_1 = \{ q \prec t \}$ 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 $\Pi = \Pi_G \cup \Pi_C$. Π_G denotes the strict rules (excluding facts) in Π, and Π_C the set of facts in Π. Let $\mathcal{P}^{\vdash}$ denote all possible literals that have a defeasible derivation from $\mathcal{P}$. Then for any $F \subseteq \Pi_C$, $(\Pi_G \cup F)^{\vdash} = (\Pi'_G \cup F)^{\vdash}$, where Π'_G follows from Π_G by **UNFOLD**$_{neg}$ (i.e., $\Pi_G \mapsto_{Uneg} \Pi'_G$).*

Proof. (Sketch)[8] 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 $\Pi_G \mapsto_U \Pi'_G$ (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 viceversa. 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 $r_i = A \leftarrow \mathcal{B}$, and let $r_{i+1} = G \leftarrow \mathcal{B}'$. 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_1 by $A \leftarrow (\mathcal{B} - \{G\} \cup \mathcal{B}')$. 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**$_{neg}(\Pi_G \cup F)$. The proof is analogous the other way around.

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

Proof. Assume $\langle \mathcal{A}, H\rangle$ is an argument. Then $\Pi \cup \mathcal{A} \vdash H$, or equivalently $\Pi_G \cup \Pi_C \cup \mathcal{A} \vdash H$. But from Proposition 4.7 this is equivalent to $\Pi' \cup \mathcal{A} \vdash H$, where $\Pi_G \cup \Pi_C \mapsto_{Uneg} \Pi'$. Clearly, this defeasible derivation is non-contradictory, and minimal. Hence $\langle \mathcal{A}, H\rangle \in Args(\mathcal{P}')$

Corollary 4.9. *Let $\mathcal{P}$' be the program resulting of applying **UNFOLD**$_{neg}$, i.e. $\mathcal{P} \mapsto_{Uneg} \mathcal{P}'$. Then $SEM_{\mathrm{DeLP}}(\mathcal{P}) = SEM_{\mathrm{DeLP}}(\mathcal{P}')$.*

Let us now consider tautology elimination.

Proposition 4.10. *Let $\mathcal{P}$ be a DeLP$_{neg}$ program, and $\mathcal{P}'$ the program resulting from applying **TAUT**$_{neg}$ to $\mathcal{P}$, i.e. $\mathcal{P} \mapsto_T \mathcal{P}'$ Then $SEM_{\mathrm{DeLP}}(\mathcal{P}) = SEM_{\mathrm{DeLP}}(\mathcal{P}')$.*

[8] For space reasons we do not include the proof in detail; the interested reader is referred to [CDSS00].

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

4.2 Transformation Properties in $\boldsymbol{DeLP_{not}}$

$DeLP_{not}$ can be seen as *NLP* with the addition of defeasible rules. In such a setting there is no strict negation "$\sim$", and therefore no contradictory literals P and $\sim P$. The attack relationship among arguments is completely captured by the semantics of default negation in *DeLP*: $notH$ holds iff H cannot be *justified* [GSC98]. In this respect, *DeLP* naturally extends the intended meaning of default negation in traditional logic programming ($notH$ holds iff H fails to be finitely proven).[9] Since a $DeLP_{not}$ program does not involve strict negation, many problems considered in Subsection 4.1 do not arise. New transformations $\mathbf{RED}^+_{not}$, $\mathbf{RED}^-_{not}$, $\mathbf{SUB}_{not}$, $\mathbf{UNFOLD}_{not}$ and $\mathbf{TAUT}_{not}$ 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 $DeLP_{not}$ 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. ($\mathbf{RED}^+_{not}$): Let $\langle \mathcal{A}, H\rangle$ be an argument in $\mathcal{P}$, such that $r{=}P \leftarrow Q_1, \ldots, notQ, \ldots, Q_k$ is a strict rule used in the defeasible derivation of H in $\mathcal{A}$. The literal $notQ$ holds iff Q is not justified. Since there is no rule with head Q in $\mathcal{P}$, there is no argument for Q, and hence no justification for Q. Therefore $\langle \mathcal{A}, H\rangle$ is also an argument in $\mathcal{P} - \{r\} \cup \{r'\}$, 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 $\mathcal{P} \mapsto_{Nneg} \mathcal{P}'$, then $\mathrm{SEM}_{DeLP}(\mathcal{P}){=}\mathrm{SEM}_{DeLP}(\mathcal{P}')$.
2. ($\mathbf{RED}^-_{not}$): Let $r = P \leftarrow Q_1, \ldots, notQ, \ldots Q_n$ be a strict rule, and assume $Q \leftarrow\ \in \mathcal{P}$. If r is used in a defeasible derivation for building an argument $\langle \mathcal{A}, H\rangle$, the literal $notQ$ will hold iff Q is not justified. But the empty argument $\langle \emptyset, Q\rangle$ is a justification for Q. Hence r cannot be used in any argument, so that any argument $\langle \mathcal{A}, H\rangle$ in $Args(\mathcal{P})$ is also an argument in $Args(\mathcal{P}-\{r\})$. The same applies for any other strict or defeasible rule used in the defeasible derivation of H in $\mathcal{A}$. Therefore if $\mathcal{P} \mapsto_{Pneg} \mathcal{P}'$, then $\mathrm{SEM}_{DeLP}(\mathcal{P}){=}\mathrm{SEM}_{DeLP}(\mathcal{P}')$.
3. ($\mathbf{SUB}_{not}$): Let $\mathcal{P}$ be a $DeLP_{not}$ program, and let $r = P \leftarrow \mathcal{B}_1$ be a non-minimal rule in $\mathcal{P}$ (i.e., there exists a rule $r' = P \leftarrow \mathcal{B}_2$ such that $\mathcal{B}_2 \subseteq \mathcal{B}_1$). If there is an argument $\mathcal{A}$ for H using rule r in the defeasible derivation of H, then the same argument $\mathcal{A}$ for H can be built by using r' instead (since every literal $notQ$ that holds in $\mathcal{B}_2$ also holds in $\mathcal{B}_1$). Hence $Args(\mathcal{P}) = Args(\mathcal{P} - \{r\})$. The same applies for any other strict rule used in the defeasible derivation of h in $\mathcal{A}$. Therefore $\mathbf{SUB}_{not}$ holds wrt strict rules.
4. ($\mathbf{UNFOLD}_{not}$): this transformation holds in $DeLP_{not}$, following the same line of reasoning used in Proposition 4.8.
5. ($\mathbf{TAUT}_{not}$): tautology elimination holds in $DeLP_{not}$, following the same line of reasoning used in Proposition 4.10.

[9] Note that default negation is applied to positive literals (i.e., atoms) in *NLP*, whereas in *DeLP* it can be applied to arbitrary literals.

	NLP under wfs	$DeLP_{neg}$	$DeLP_{not}$
RED$^+$	yes	no	yes
RED$^-$	yes	yes	yes
SUB	yes	yes, for strict rules	yes, for strict rules
UNFOLD	yes	yes[a], for strict rules	yes[a], for strict rules
TAUT	yes	yes	yes

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

[a] Some additional conditions are required for the transformation to hold.

5 Relating *NLP* and *DeLP*

Figure 1 summarizes the behavior of *NLP*, $DeLP_{neg}$ and $DeLP_{not}$ 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 *notP* 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 *not* L should too (similarly, if L then *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 $\mathbf{RED}^-$ 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

[ADP95] J. J. Alferes, Carlos Viegas Damasio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Journal of Automated Reasoning*, 14(1):93–147, 1995.

[BD98] S. Brass and J. Dix. Characterizations of the Disjunctive Well-founded Semantics: Confluent Calculi and Iterated GCWA. *Journal of Automated Reasoning*, 20(1):143–165, 1998.

[BD99] S. Brass and J. Dix. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, 38(3):167–213, 1999.

[BD01] Gerhard Brewka and Jürgen Dix. Knowledge representation with extended logic programs. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition, Volume 6, Methodologies*, chapter 6. Reidel Publ., 2001. Shortened version also appeared in Dix, Pereira, Przymusinski (Eds.), *Logic Programming and Knowledge Representation*, Springer LNAI 1471, pages 1–55, 1998.

[BDFZ01] Stefan Brass, Juergen Dix, Burkhard Freitag, and Ulrich Zukowski. Transformation-based Bottom-up Computation of the Well-Founded Model. *Theory and Practice of Logic Programming*, 2001. to appear.

[CDSS00] C.I. Chesñevar, J. Dix, F. Stolzenburg, and G. Simari. Defeasible Logic Programs under Brass-Dix' transformations. Technical report, Dept. of CS, Universidad Nacional del Sur, Av. Alem 1253, February 2000.

[CS99] Carlos I. Chesñevar and Guillermo R. Simari. Modeling defeasibility into an extended logic programming setting using an abstract argumentation framework. In *Proceedings of the Argentinean Symposium on Artificial Intelligence*, pages 71–89. JAIIO, Buenos Aires, Argentina, September 1999.

[DPP97] J. Dix, L. Pereira, and T. Przymusinski. Prolegomena to logic programming for non-monotonic reasoning. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming - LNAI 1216*, pages 1–36. Springer Verlag, 1997.

[Dun93] Phan M. Dung. An argumentation semantics for logic programming with explicit negation. In *Proc. ICLP'93*, pages 616–630. MIT Press, 1993.

[Gar97] Alejandro J. Garcia. *Defeasible Logic Programming: Definition and Implementation* (MSc Thesis). Departamento de Cs. de la Computacion, Universidad Nacional del Sur, Bahía Blanca, Argentina, July 1997.

[GL90] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In D. Warren and P. Szeredi, editors, *Proc. ICLP*, pages 579–597. MIT Press, 1990.

[GSC98] Alejandro J. Garcia, Guillermo R. Simari, and Carlos I. Chesñevar. An argumentative framework for reasoning with inconsistent and incomplete information. In *Workshop on Practical Reasoning and Rationality.* 13th biennial European Conference on Artificial Intelligence (ECAI-98), August 1998.

[KT99] Antonios C. Kakas and Francesca Toni. Computing argumentation in logic programming. *Journal of Logic and Computation*, 9(4):515–562, 1999.

[Lif94] V. Lifschitz. *Circmscription*, pages 297–353. Clarendon, Oxford, 1994.

[PS97] Henry Prakken and Giovanni Sartor. Argument-based logic programming with defeasible priorities. *Journal of Applied Non-classical Logics*, 7:25–75, 1997.

[SCG94] Guillermo R. Simari, Carlos I. Chesñevar, and Alejandro J. Garcia. The role of dialectics in defeasible argumentation. In *Anales de la XIV Conferencia Internacional de la Sociedad Chilena para Ciencias de la Computacion.* Concepcion, Chile, November 1994.

[SL92] Guillermo R. Simari and Ronald P. Loui. A Mathematical Treatment of Defeasible Reasoning and its Implementation. *Artificial Intelligence*, 53:125–157, 1992.