Relating Defeasible and Normal Logic Programming through Transformation Properties ⋆

Carlos Iván Chesñevar

Univ. Nacional del Sur, Av. Alem 1253, (8000) Bahía Blanca, ARGENTINA
cic@cs.uns.edu.ar

Jürgen Dix

Dept. of Computer Science, The University of Manchester,
Oxford Road, Manchester M13 9PL, UNITED KINGDOM
dix@cs.man.ac.uk

Frieder Stolzenburg

Institut für Informatik, Universität Koblenz-Landau
Rheinau 1, 56075 Koblenz, GERMANY
stolzen@uni-koblenz.de

Guillermo Ricardo Simari

Univ. Nacional del Sur, Av. Alem 1253, (8000) Bahía Blanca, ARGENTINA
grs@cs.uns.edu.ar

Abstract

This paper relates the *Defeasible Logic Programming* (*DeLP*) framework and its semantics SEM_{DeLP} to classical logic programming frameworks. In *DeLP* we distinguish between two different sorts of rules: *strict* and *defeasible* rules. Negative literals ($\sim A$) in these rules are considered to represent *classical* negation. In contrast to this, in *normal logic programming* (*NLP*), there is only one kind of rules, but the meaning of negative literals ($\mathsf{not}\, A$) is different: they represent a kind of *negation as failure*, and thereby introduce defeasibility. Various semantics have been defined for *NLP*, notably the well-founded semantics (WFS) in [VGRS88,vRS91] and the stable semantics Stable in [GL88,GL91].

In this paper we consider the *transformation properties* for *NLP* introduced by Brass and Dix and suitably adjusted for the *DeLP* framework. We show which transformation properties are satisfied, thereby identifying aspects in which *NLP* and *DeLP* differ. We contend that the 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*.

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

---

1 Introduction and motivations

Defeasible Logic Programming (*DeLP*) [SL92,Gar97,GSC98] is a logic programming formalism which relies upon *defeasible argumentation* [PV01,CML00] for solving queries. *DeLP* combines *strict rules*, defined as in extended logic programming, and *defeasible rules*, of the form $A \prec\!\!\!- B$, indicating that reasons to believe in the antecedent B provide reasons to believe in the consequent A. Solving a query Q in *DeLP* gives rise to a proof $\mathcal{A}$ for Q (written $\langle \mathcal{A}, Q \rangle$ for short) involving both strict and defeasible rules, called *argument*. In order to determine whether Q is ultimately accepted as justified belief, a recursive analysis is performed which involves finding *defeaters*, i.e., arguments against accepting $\mathcal{A}$, which are better than $\mathcal{A}$ (according to a preference criterion). Since defeaters are arguments, a recursive procedure is to be carried out, in which defeaters, defeaters of defeaters, and so on, must be taken into account.

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 and stable model semantics: we refer to [DPP97,BD02,DFN01] for an in-depth discussion of extensions of logic programming and their semantics. In contrast, *DeLP* has an "operational" semantics which is determined by the outcome of the dialectical process used for answering queries.

In [BD97,BD98,BD99], a number of *transformation rules* were introduced which allow to "simplify" a normal logic program (*nlp*) P to get its well-founded semantics 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

⋆ 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 [DSSF99]. A preliminary version appeared at CACIC '2000 [CDSS00].

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*, analizing two classes of *DeLP* programs: $DeLP_{neg}$ (Subsection 4.1) and $DeLP_{not}$ (Subsection 4.2). Subsection 4.4 summarizes the relationships between *NLP* and *DeLP*, and the main results we have obtained. Finally, Section 5 discusses related work and concludes.

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 because, following [GL88,Lif94], program rules with variables can be viewed as "schemata" that represent their ground instances. Although there now exist powerful grounding techniques applied by various implementations (smodels, DLV) we believe that handlind programs with free variables and computing appropriate substitutions (variable bindings) can often improve the performance of the system. Therefore, whenever suitable, we are also using the formalism of *most general unifiers* (mgU) stemming from logic programming.

2.1 Normal Logic Programs (NLP)

Definition 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 $A \leftarrow L_1, \ldots, L_k$, where A is an atom and each L_i is an atom B or its negation $\mathsf{not}\, B$. If $\mathcal{B} = \{L_1, \ldots, L_k\}$ is the body of a rule $A \leftarrow \mathcal{B}$, we also use the notation $A \leftarrow \mathcal{B}^+, \mathsf{not}\, \mathcal{B}^-$, where $\mathcal{B}^+$ (resp. $\mathcal{B}^-$) contains all the positive (resp. negative) body atoms in $\mathcal{B}$.*

In *NLP*, atoms A and negated atoms $\mathsf{not}\, A$ 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*, referring to an atom A and its default negation $\mathsf{not}\, A$. 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 $\text{Prog}_{\mathcal{L}}$ 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 $I_1 \cup \mathsf{not}\, I_2$.

Definition 2 (Semantics SEM) *A semantics SEM is a mapping which assigns to each 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 each program $\mathcal{P}$ the partial interpretation SEM($\mathcal{P}$) is a 3-valued model*[1] *of $\mathcal{P}$.*

In Section 3 we will consider a particular 3-valued semantics for the class *NLP* called the wellfounded semantics 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 "$\sim$" is called *strong negation*. In addition, we will consider *default negation* with "not" here. We define formally:

Definition 3 (Literal, assumption literal) *A* literal *L is an atom p or a negated atom $\sim p$, where the symbol "$\sim$" represents* strong negation. *An* assumption literal *A has the form "$\mathsf{not}\, A$", where A is a literal.*

Definition 4 (Strict rules *$Head \leftarrow Body$***)** *A* strict rule *is an ordered pair, conveniently denoted as $Head \leftarrow Body$, the first member of which, $Head$, is a literal, and the second member, $Body$, is a finite set of literals, which may be (additionally) negated with "*not*" (default negation). A strict rule with the head L_0 and body $\{L_1, \ldots, L_k\}$ can also be written as $L_0 \leftarrow L_1, \ldots, L_k$. If the body is empty, it is written $L \leftarrow true$, and it is called a* fact. *Facts may also be written as L.*

Definition 5 (Defeasible rules *$Head \mathrel{-\!\!<} Body$***)** *A* defeasible rule *is an*

[1] We equip $\leftarrow$ with the Kleene interpretation, where undef $\leftarrow$ undef is considered to be true.

ordered pair, conveniently denoted as Head $-\!\!<$ *Body, the first member of which, Head, is a literal, and the second member, Body, is a finite set of literals, which may be (additionally) negated with "*not*". A defeasible rule with the head* L_0 *and body* $\{L_1, \ldots, L_k\}$ *can also be written as* $L_0 -\!\!< L_1, \ldots, L_k$*,* $k > 0$*.*

Syntactically, the symbol " $-\!\!<$ " is all that distinguishes a defeasible rule from a strict rule. Defeasible rules account for tentative information that can be used if nothing can be argued against it, whereas strict rules are used to represent non-defeasible information.

In the sequel, atoms will be denoted with lowercase letters (a, b, ...). The letter r (possibly indexed) will be used for denoting rule names. Literals will be denoted with capital letters (A, B, ...), possibly indexed. Sets of atoms will be denoted as $\mathcal{A}$, $\mathcal{B}$, ..., possibly indexed. Logic programs will be usually denoted as $\mathcal{P}_1$, $\mathcal{P}_2$, etc.

Definition 6 (Defeasible logic program $\mathcal{P} = (\Pi, \Delta)$) *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 $DeLP_{neg}$ (resp. $DeLP_{\mathsf{not}}$). Note that strong negation "$\sim$" is applied to atoms (also in rule heads), whereas default negation is applied to literals (possibly strongly negated). But default negation does not occur in heads of programs (see Definition 1). We will associate with every program $\mathcal{P}$ a set of *assumable facts* of the form $\mathsf{assume}\, L$, for every literal L in $\mathcal{P}$. Those literals will be given a special meaning in the argumentation framework. They will be used to define the semantics of default negation.

We will write $\overline{P}$ to denote the *complement* of a literal P, defined as follows: $\overline{P} =_{def} \sim P$, $\overline{\sim P} =_{def} P$, and $\overline{\mathsf{assume}\, P} =_{def} \overline{P}$.

Next we will define the notion of a *defeasible derivation* for a *dlp*. In brief, it is a finite set of rules obtained by backward chaining from a literal 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. The definition is similar to the one of SLDNF-derivation in [Llo87], except that literals negated with "not" are associated with assumable facts.

Definition 7 (Derivation sequence) *A defeasible derivation for a literal* Q *in a general dlp* $\mathcal{P}$ *(possibly containing assumable facts) is a finite sequence of (instantiations of) rules in* $\mathcal{P}$*. For this, we consider sequences* G_i *of goals i.e., sequences of sequences of literals, and* r_i *of rules for* $i \geq 0$ *as follows:*

(1) $G_0 = [Q]$. r_0 *is not defined.*

(2) Let $G_i = [Q_1, \ldots, Q_m, \ldots, Q_n]$ *with* $1 \leq m \leq n$.

- *If there is a strict or defeasible rule in* $\mathcal{P}$ *with head* L_0 *and body* $\{L_1, \ldots, L_k\}$ *such that* L_0 *and* Q_m *have the most general unifier* σ*, then* $G_{i+1} = [Q_1, \ldots, Q_{m-1}, L_1, \ldots, L_k, Q_{m+1}, \ldots, Q_n]\sigma$ *and* $r_{i+1} = (L_0 \leftarrow L_1, \ldots, L_k)\sigma$ *or* $r_{i+1} = (L_0 \prec\!\!\!- L_1, \ldots, L_k)\sigma$*, respectively.*
- *If* Q_m *has the form* $\mathsf{not}\, L$ *for some literal* L *(possibly negated with* $\sim$*) and the assumable fact* $r = \mathsf{assume}\, \overline{L}$ *is in* $\mathcal{P}$*, then* $G_{i+1} = [Q_1, \ldots, Q_{m-1}, Q_{m+1}, \ldots, Q_n]$ *and* $r_{i+1} = r$.

The sequence of rules $S = [r_1, \ldots, r_l]$ *(for some suitable* $l > 0$*) is called* defeasible derivation *for* Q *in* $\mathcal{P}$ *iff the corresponding sequence* G_l *is empty. We say that* Q *can be defeasibly derived from* $\mathcal{P}$ *and write* $\mathcal{P} \vdash Q$ *in this case.*

Definition 8 (Contradictory set of rules) *A set of rules* $\mathcal{S}$ *is* contradictory *iff there is a defeasible derivation from* $\mathcal{S}$ *for some literal* P *and its complement* $\overline{P}$*, i.e.,* $\mathcal{S} \vdash P$ *and* $\mathcal{S} \vdash \overline{P}$.

Given a *dlp* $\mathcal{P}$, we will always assume that the set Π of strict rules is non-contradictory (i.e., there is no literal P such that $\Pi \vdash P$ and $\Pi \vdash \sim P$). If a contradictory set of strict rules were used in a *dlp*, the same problems as in extended logic programming would appear. The corresponding analysis has been done elsewhere [GL90].

Example 9 *Consider an engine the performance of which is determined by two switches* $sw1$ *and* $sw2$*. The switches regulate different features of the engine's behavior, such as pumping system and working speed. We can model the engine behavior using a dlp program* (Π, Δ)*, where*

$$\Pi = \{(sw1 \leftarrow), (sw2 \leftarrow), (heat \leftarrow), (\sim fuel_ok \leftarrow pump_clogged)\}$$

(specifying that the two switches are on, there is heat, and whenever the pump gets clogged, fuel is not ok), and Δ *models the possible behavior of the engine under different conditions (Figure 1).*

Next we introduce the definition of argument in *DeLP*. Basically, an argument for a literal Q is a defeasible derivation $S = [r_1, \ldots, r_k]$ which is non-contradictory with respect to a given *dlp* program, and the defeasible information in S is minimal with respect to set inclusion.

Definition 10 (Argument) *Given a dlp* $\mathcal{P} = (\Pi, \Delta)$*, we will define* $\mathcal{HB}_{\mathsf{ass}} = \{\mathsf{assume}\, L \mid L \text{ is a literal in } \mathcal{P}\}$*. An* argument $\mathcal{A}$ *for a query* Q*, denoted* $\langle \mathcal{A}, Q \rangle$*, is defined as* $\mathcal{R}_A \cup \mathcal{H}_A$*, where* $\mathcal{R}_A$ *is a subset of ground instances of the defeasible rules of* $\mathcal{P}$ *and* $\mathcal{H}_A \subseteq \mathcal{HB}_{\mathsf{ass}}$*, such that:*

(1) there exists a defeasible derivation for Q *from* $\Pi \cup \mathcal{A}$.

$pump_fuel_ok \multimap sw1$

(when sw1 is on, normally fuel is pumped properly);

$fuel_ok \multimap pump_fuel_ok$

(when fuel is pumped, normally fuel works ok);

$pump_oil_ok \multimap sw2$

(when sw2 is on, normally oil is pumped);

$oil_ok \multimap pump_oil_ok$

(when oil is pumped, normally oil works ok);

$engine_ok \multimap fuel_ok, oil_ok$

(when there is fuel and oil, normally engine works ok);

$\sim engine_ok \multimap fuel_ok, oil_ok, heat$

(when there is fuel, oil and heat, usually engine is not working ok);

$pump_clogged \multimap pump_fuel_ok, low_speed$

(when fuel is pumped and speed is low, there are

reasons to believe that the pump is clogged);

$low_speed \multimap sw2$

(when sw2 is on, normally speed is low);

$\sim low_speed \multimap sw2, sw1$

(when both sw2 and sw1 are on, speed is considered not to be low).

Fig. 1. Set Δ (Example 9)

(2) $\Pi \cup \mathcal{A}$ *is non-contradictory, and*
(3) $\mathcal{A}$ *is minimal with respect to set inclusion.*

An argument $\langle \mathcal{A}, Q \rangle$ is *strict* iff $\mathcal{A} = \emptyset$. An argument $\langle \mathcal{A}_1, Q_1 \rangle$ is a *subargument* of another argument $\langle \mathcal{A}_2, Q_2 \rangle$, if $\mathcal{A}_1 \subseteq \mathcal{A}_2$. Given an argument $\langle \mathcal{A}, Q \rangle$, we will also write $\mathcal{H}_{\langle \mathcal{A},Q \rangle}$ to denote the set of assumption literals in $\langle \mathcal{A}, Q \rangle$. Next we introduce the auxiliary notion of *immediate subargument*, which will be used later in the proofs of Propositions 48 and 60.

Definition 11 (Immediate subarguments) *Let $\langle \mathcal{A}, H \rangle$ be an argument, such that $H \leftarrow P_1, \ldots, P_k$ is the last strict rule used in the defeasible derivation of H from $\Pi \cup \mathcal{A}$. Clearly, in such a case there exist subsets $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of $\mathcal{A}$, which are arguments for $P_1, \ldots, P_k$. We will call $\langle \mathcal{A}_1, P_1 \rangle$, ... , $\langle \mathcal{A}_k, P_k \rangle$* immediate subarguments *of $\langle \mathcal{A}, H \rangle$.*

Example 12 *Consider the dlp program* (Π, Δ)*, with*

$$\begin{aligned} \Pi &= \{(p \leftarrow q, \mathsf{not}\, r), (w \leftarrow q, r), (s \leftarrow)\} \\ \Delta &= \{(q \prec s), (r \prec s)\} \end{aligned}$$

It follows that $\mathcal{A} = \{(q \prec s), (r \prec s)\}$ *is an argument for* w*, and* $\mathcal{B} = \{\mathsf{assume} \sim r, (q \prec s)\}$ *is an argument for* p*. In the argument* $\langle \mathcal{B}, p\rangle$ *the last strict rule used in the derivation of* p *is* $p \leftarrow q, \mathsf{not}\, r$*. Then* $\mathcal{B}' = \{q \prec s\}$ *is an argument for* q*, and it is an immediate subargument of* $\langle \mathcal{B}, p\rangle$*. In the argument* $\langle \mathcal{A}, w\rangle$ *the last strict rule used in the derivation of* w *is* $w \leftarrow q, r$*. Then* $\langle \mathcal{A}, q\rangle$ *and* $\langle \mathcal{A}, r\rangle$ *are immediate subarguments of* $\langle \mathcal{A}, w\rangle$*.*

Example 13 *Consider Example 9. Then the set*

$$\begin{aligned} \mathcal{A} = \{ & (pump_fuel_ok \prec sw1),\ (pump_oil_ok \prec sw2), \\ & (fuel_ok \prec pump_fuel_ok),\ (oil_ok \prec pump_oil_ok), \\ & (engine_ok \prec fuel_ok, oil_ok)\ \} \end{aligned}$$

is an argument for $engine_ok$*. The set*

$$\begin{aligned} \mathcal{B} = \{ & (pump_fuel_ok \prec sw1),\ (low_speed \prec sw2), \\ & (pump_clogged \prec pump_fuel_ok, low_speed)\ \} \end{aligned}$$

is an argument for $\sim fuel_ok$*. The set* $\mathcal{C} = \{\sim low_speed \prec sw2, sw1\}$ *is an argument for* $\sim low_speed$*.*

Given a *dlp* program $\mathcal{P}$, we will denote by $Args(\mathcal{P})$ the set of all possible arguments $\langle \mathcal{A}, Q\rangle$ that can be built from $\mathcal{P}$ wrt. arbitrary queries Q. We emphasize that this set consists of pairs $\langle \mathcal{A}, Q\rangle$ and not just of arguments $\mathcal{A}$ alone. This makes the condition $Args(\mathcal{P}) = Args(\mathcal{P})'$ much stronger and is important for our Propostion 37 to hold.

The following definition captures the notion of conflict between two arguments.

Definition 14 (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* $\Pi \cup \{Q_1, Q\}$ *is contradictory.*

Example 15 *Consider Example 13. Then* $\langle \mathcal{B}, \sim fuel_ok\rangle$ *is a counterargument for* $\langle \mathcal{A}, engine_ok\rangle$*, since there is a subargument* $\mathcal{A}' = \{\ fuel_ok \prec pump_fuel_ok,\ oil_ok \prec pump_oil_ok,\ engine_ok \prec fuel_ok, oil_ok\ \}$ *for* $fuel_ok$*, such that* $\Pi \cup \{fuel_ok, \sim fuel_ok)$ *is contradictory.*

Informally, a query Q will succeed if the supporting argument is not defeated; that argument becomes a *justification*. In order to establish that $\mathcal{A}$ 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. *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. Next we will introduce this concept formally.

Definition 16 (Specificity) *Given a dlp program $\mathcal{P}$, let Π_G denote the set of all rules with nonempty bodies. Let Fdenote the set of all possible literals that have a defeasible derivation in $\mathcal{P}$.*

An argument $\langle \mathcal{A}_1, Q_1 \rangle$ is strictly more specific *than an argument $\langle \mathcal{A}_2, Q_2 \rangle$ (denoted $\langle \mathcal{A}_1, Q_1 \rangle \succ \langle \mathcal{A}_2, Q_2 \rangle$) if and only if:*

(1) For all $H \subseteq F$: if $\Pi_G \cup H \cup \mathcal{A}_1 \vdash Q_1$ and $\Pi_G \cup H \nvdash Q_1$, then $\Pi_G \cup H \cup \mathcal{A}_2 \vdash Q_2$.
(2) There exists $H' \subseteq F$ such that $\Pi_G \cup H' \cup \mathcal{A}_2 \vdash Q_2$ and $\Pi_G \cup H' \nvdash Q_2$ and $\Pi_G \cup H' \cup \mathcal{A}_1 \nvdash Q_1$.

Example 17 *Consider the following dlp $\mathcal{P}$:*

$$\mathcal{P} = \{ (p \prec f_1, f_2), (\sim p \prec f_1), (f_1 \leftarrow), (f_2 \leftarrow) \}$$

Then the set of all literals derivable in $\mathcal{P}$ is $F = \{ p, \sim p, f_1, f_2 \}$. Consider the arguments $\langle \mathcal{A}_1, p \rangle$ and $\langle \mathcal{A}_2, \sim p \rangle$, with $\mathcal{A}_1 = \{ p \prec f_1, f_2 \}$ and $\mathcal{A}_2 = \{ \sim p \prec f_1 \}$. For every $H \subseteq F$, condition 1 in Definition 16 holds. For $H' = \{f_1\}$, condition 2 in Definition 16 holds. Hence $\langle \mathcal{A}_1, p \rangle$ is strictly more specific than $\langle \mathcal{A}_2, \sim p \rangle$.

Definition 18 (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 strictly more specific than $\langle \mathcal{A}, Q \rangle$. In this case $\langle \mathcal{A}_1, Q_1 \rangle$ is called a proper defeater *of $\langle \mathcal{A}, Q \rangle$, or*
(b) Neither $\langle \mathcal{A}_1, Q_1 \rangle$ is strictly more specific than $\langle \mathcal{A}_2, Q_2 \rangle$, nor $\langle \mathcal{A}_2, Q_2 \rangle$ is strictly more specific than $\langle \mathcal{A}_1, Q_1 \rangle$. In this case $\langle \mathcal{A}_1, Q_1 \rangle$ is a blocking defeater *of $\langle \mathcal{A}, Q \rangle$.*

Example 19 *Consider Examples 13 and 15. Then $\langle \mathcal{B}, \sim fuel_ok \rangle$ is a proper defeater for $\langle \mathcal{A}, engine_ok \rangle$, since it is more specific.*

This conceptualization allows us to apply the notion of counterargumentation (Definition 14) and defeat (Definition 18) in a natural way when assumption

literals are involved, as shown in the following example.

Example 20 *Consider a dlp $\mathcal{P}$ = (Π, Δ), where*

$$\begin{aligned} \Pi &= \{ r \leftarrow , s \leftarrow , t \leftarrow , q \leftarrow s \}, \\ \Delta &= \{ p \prec \mathsf{not}\, q, r,\ q \prec t \} \end{aligned}$$

Then $\mathcal{A}$ =$\{ p \prec \mathsf{not}\, q, r,\ \mathsf{assume} \sim q \}$ is an argument for p, which is counterargued by the argument $\langle \{q \prec t\}, q \rangle$ as well as by the argument $\langle \emptyset, q \rangle$.

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*. The formal definitions required for this process are as follows:

Definition 21 (Argumentation line) *Let $\mathcal{P}$ be a dlp, and let $\langle \mathcal{A}, Q \rangle$ be an argument in $\mathcal{P}$. An* argumentation line *starting from $\langle \mathcal{A}, Q \rangle$, denoted $\lambda^{\langle \mathcal{A}, Q \rangle}$ (or simply λ) is a possibly infinite sequence of arguments*

$$\lambda^{\langle \mathcal{A}, Q \rangle} = [\, \langle \mathcal{A}_0, Q_0 \rangle, \langle \mathcal{A}_1, Q_1 \rangle, \langle \mathcal{A}_2, Q_2 \rangle, \ldots, \langle \mathcal{A}_n, Q_n \rangle \ldots]$$

satisfying the following conditions:

(1) If $\langle \mathcal{A}, Q \rangle$ has no defeaters, then $\lambda^{\langle \mathcal{A}, Q \rangle} = [\, \langle \mathcal{A}, Q \rangle \,]$.
(2) If $\langle \mathcal{A}, Q \rangle$ has a defeater $\langle \mathcal{B}, S \rangle$ in $\mathcal{P}$, then $\lambda^{\langle \mathcal{A}, Q \rangle} = \langle \mathcal{A}, Q \rangle \circ \lambda^{\langle \mathcal{B}, S \rangle}$.

We distinguish two sets in any argumentation line λ: the set of supporting arguments $\lambda_S = \{ \langle \mathcal{A}_0, Q_0 \rangle, \langle \mathcal{A}_2, Q_2 \rangle, \langle \mathcal{A}_4, Q_4 \rangle, \ldots \}$ *and the set of* interferring arguments $\lambda_I = \{ \langle \mathcal{A}_1, Q_1 \rangle, \langle \mathcal{A}_3, Q_3 \rangle, \langle \mathcal{A}_5, Q_5 \rangle, \ldots \}$.

Argumentation lines can be thought of as an exchange of arguments between two parties, a proponent and an opponent [Res77]. Dialectics imposes additional requirements on such an argument exchange to be considered rationally acceptable. In such a setting, *fallacious* reasoning (such as circular argumentation and falling into self-contradiction) is to be avoided. This can be done by requiring that all argumentation lines be *acceptable* [SCG94]. An acceptable argumentation line starting with an argument $\langle \mathcal{A}_0, Q_0 \rangle$ constitutes an exchange of arguments which can be pursued until no more arguments can be introduced because of the dialectical constraints discussed above. These notions will be introduced in the following definitions.

Definition 22 (Contradictory set of arguments) *Given a dlp $\mathcal{P} = (\Pi, \Delta)$, a set of arguments $S = \bigcup_{i=1}^{n} \{\langle \mathcal{A}_i, Q_i \rangle\}$ is* contradictory wrt $\mathcal{P}$ *iff $\Pi \cup \bigcup_{i=1}^{n} \mathcal{A}_i$ is contradictory.*

Definition 23 (Acceptable argumentation line) *Let $\mathcal{P}$ be a dlp, and let $\lambda = [\langle \mathcal{A}_0, Q_0\rangle, \langle \mathcal{A}_1, Q_1\rangle, \ldots, \langle \mathcal{A}_n, Q_n\rangle, \ldots]$ be an argumentation line in $\mathcal{P}$. Let $\lambda' = [\langle \mathcal{A}_0, Q_0\rangle, \langle \mathcal{A}_1, Q_1\rangle, \ldots, \langle \mathcal{A}_k, Q_k\rangle, \ldots]$ be an initial segment of λ. The sequence λ' is an* acceptable argumentation line *in $\mathcal{P}$ iff it is the longest initial segment in λ satisfying the following conditions:*

(1) The sets ${\lambda'}_S$ and λ'_I are each non-contradictory sets of arguments wrt $\mathcal{P}$.
(2) No argument $\langle \mathcal{A}_j, Q_j\rangle$ in λ' is a sub-argument of an earlier argument $\langle \mathcal{A}_i, Q_i\rangle$ of λ' $(i < j)$.
(3) There is no subsequence of arguments $[\langle \mathcal{A}_{i-1}, Q_{i-1}\rangle, \langle \mathcal{A}_i, Q_i\rangle, \langle \mathcal{A}_{i+1}, Q_{i+1}\rangle]$ in λ', such that $\langle \mathcal{A}_i, Q_i\rangle$, is a blocking defeater for $\langle \mathcal{A}_{i-1}, Q_{i-1}\rangle$ and $\langle \mathcal{A}_{i+1}, Q_{i+1}\rangle$ is a blocking defeater for $\langle \mathcal{A}_i, Q_i\rangle$.

The rationale for the conditions in Definition 23 can be better understood in a dialectical setting [SCG94]. Condition 1 disallows the use of contradictory information on either side (proponent or opponent). Condition 2 eliminates the *"circulus in demonstrando"* fallacy (circular reasoning). Finally, condition 3 enforces the use of a stronger argument to defeat an argument which acts as a blocking defeater.

Example 24 *Consider Example 9. The sequence*

$$\lambda_1 = [\ \langle \mathcal{A}, engine_ok\rangle,\ \langle \mathcal{B}, \sim fuel_ok\rangle,\ \langle \mathcal{C}, \sim low_speed\rangle\ ]$$

is an acceptable argumentation line*, whereas any sequence having the initial segment*

$$\lambda_2 = [\langle \mathcal{A}, engine_ok\rangle,\ \langle \mathcal{B}, \sim fuel_ok\rangle,\ \langle \mathcal{D}, fuel_ok\rangle\ ]$$

with $\mathcal{D} = \{\ pump_fuel_ok \prec sw1,\ fuel_ok \prec pump_fuel_ok\ \}$ is an argumentation line which is not acceptable, since the last argument defeats $\langle \mathcal{B}, \sim fuel_ok\rangle$, but it is a subargument of a previous argument in λ_2 (viz. $\langle \mathcal{A}, engine_ok\rangle$). Hence $\langle \mathcal{D}, fuel_ok\rangle$ is deemed as a fallacious argument to be excluded from the dialectical analysis.

Proposition 25 *Any acceptable argumentation line in a dlp $\mathcal{P}$ is finite.*

PROOF. Since $\mathcal{P}$ has no function symbols, and $\mathcal{P}$ is a finite set of program rules, the set of all possible arguments $Args(\mathcal{P})$ is necessarily finite. Hence the only way to get an infinite argumentation line $\lambda = [\ \langle \mathcal{A}_0, Q_0\rangle, \langle \mathcal{A}_1, Q_1\rangle, \langle \mathcal{A}_2, Q_2\rangle, \ldots, \langle \mathcal{A}_n, Q_n\rangle\ \ldots]$ is by having the same argument twice in λ, i.e., $\langle \mathcal{A}_i, Q_i\rangle = \langle \mathcal{A}_j, Q_j\rangle$, and hence $\mathcal{A}_i = \mathcal{A}_j$. But this cannot be the case in an acceptable argumentation line because of condition 2 in Definition 23. Therefore any acceptable argumentation line λ is necessarily finite.

Let $\Lambda^{\langle \mathcal{A}_0, Q_0 \rangle} = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ be the set of all acceptable argumentation lines starting with $\langle \mathcal{A}_0, Q_0 \rangle$ in a *dlp* $\mathcal{P}$. A tree structure can be built out of the elements of $\Lambda^{\langle \mathcal{A}_0, Q_0 \rangle}$, so that every path in the tree corresponds to some $\lambda_i \in \Lambda^{\langle \mathcal{A}_0, Q_0 \rangle}$. This structure will be called *dialectical tree*. Formally:

Definition 26 (Dialectical tree) *Let $\mathcal{P}$ be a dlp, and let $\mathcal{A}_0$ be an argument for Q_0 in $\mathcal{P}$. A* dialectical tree *for $\langle \mathcal{A}_0, Q_0 \rangle$, denoted $\mathcal{T}_{\langle \mathcal{A}_0, Q_0 \rangle}$, is a tree structure defined as follows:*

(1) The root node of $\mathcal{T}_{\langle \mathcal{A}_0, Q_0 \rangle}$ is $\langle \mathcal{A}_0, Q_0 \rangle$.
(2) $\langle \mathcal{B}', H' \rangle$ is an immediate child of $\langle \mathcal{B}, H \rangle$ iff there exists an acceptable argumentation line $\lambda^{\langle \mathcal{A}_0, Q_0 \rangle} = [\langle \mathcal{A}_0, Q_0 \rangle, \langle \mathcal{A}_1, Q_1 \rangle, \ldots, \langle \mathcal{A}_n, Q_n \rangle\ ]$ such that there are two elements $\langle \mathcal{A}_{i+1}, Q_{i+1} \rangle = \langle \mathcal{B}', H' \rangle$ and $\langle \mathcal{A}_i, Q_i \rangle = \langle \mathcal{B}, H \rangle$, for some $i = 0, \ldots, n-1$.

Clearly, leaves in a dialectical tree correspond to undefeated arguments. Defeat among arguments in a dialectical tree can be propagated from the leaves up to the root, according to the marking procedure given in Definition 27.

Definition 27 (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 each 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 one child marked as U-node.

An argument $\mathcal{A}$ for a literal Q which turns to be ultimately labeled as undefeated in $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$ is called a *justification* for Q.

Definition 28 (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.*

It can be shown [Gar97] that for any *dlp* $\mathcal{P}$, strict arguments in $\mathcal{P}$ have no counterarguments, and therefore no defeaters. As a direct consequence of Definitions 26, 27 and 28, it follows that any strict argument $\mathcal{A}$ for a literal Q will be a *justification* for Q: similar results hold for other argumentation systems, such as Vreeswijk's [Vre93] and Prakken and Sator's [PS97].

Example 29 *Consider Example 9, and assume our main query is engine_ok. An argument $\langle \mathcal{A}, engine_ok \rangle$ can be built, which is defeated by the argument $\langle \mathcal{B}, \sim fuel_ok \rangle$ (as shown in Examples 13, 15 and 19). Hence, the argument $\langle \mathcal{A}, engine_ok \rangle$ will be provisionally rejected, since it is defeated. However, $\langle \mathcal{A}, engine_ok \rangle$ can be* reinstated, *since there exists a third argument $\mathcal{C} = \{\sim low_speed \prec sw2, sw1\}$ for $\sim low_speed$ which in turn defeats*

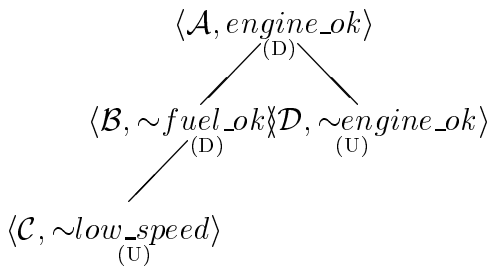

Fig. 2. Dialectical tree (Example 9)

$\langle \mathcal{B}, \sim fuel_ok \rangle$.

Hence, $\langle \mathcal{A}, engine_ok \rangle$ *comes to be undefeated again, since the argument* $\langle \mathcal{B}, \sim fuel_ok \rangle$ *was defeated. But there is another defeater for* $\langle \mathcal{A}, engine_ok \rangle$*, the argument* $\langle \mathcal{D}, \sim engine_ok \rangle$*, where* $\mathcal{D} = \{$ *pump_fuel_ok* $\prec$ *sw1, pump_oil_ok* $\prec$ *sw2, fuel_ok* $\prec$ *pump_fuel_ok, oil_ok* $\prec$ *pump_oil_ok,* $\sim$*engine_ok* $\prec$ *fuel_ok, oil_ok, heat* $\}$*. Hence* $\langle \mathcal{D}, engine_ok \rangle$ *is once again provisionally defeated.*

Since there are no more arguments to consider, $\langle \mathcal{A}, engine_ok \rangle$ *turns out to be ultimately defeated, so that we can conclude that the argument* $\langle \mathcal{A}, engine_ok \rangle$ *is not* justified.

Figure 2 shows the resulting dialectical tree, as well as its associated labeling.

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 30 (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 each 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 each 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 iff 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.*

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

Definition 31 ($\mathbf{SEM_{DeLP}}$)
For any dlp $\mathcal{P}$*, we define* $SEM_{DeLP}\ (\mathcal{P}) = \langle \mathcal{P}^{accepted}, \mathcal{P}^{rejected}, \mathcal{P}^{undef} \rangle$*, where*

$$\begin{aligned} \mathcal{P}^{accepted} &= \{Q | Q \in \mathsf{Justified}(\mathcal{P})\} \\ \mathcal{P}^{rejected} &= \{Q | Q \in \mathsf{Unknown}(\mathcal{P}) \cup \mathsf{Negative}(\mathcal{P})\} \\ \mathcal{P}^{undef} &= \{Q | Q \in \mathsf{Undefined}(\mathcal{P})\}. \end{aligned}$$

Example 32 *Consider* $\mathcal{P}$ *as defined in Example 9, and consider the analysis performed in Example 29. Then* $engine_ok \in \mathsf{Negative}(\mathcal{P})$*,* $\sim engine_ok \in \mathsf{Positive}(\mathcal{P})$*,* $heat \in \mathsf{Positive}(\mathcal{P})$*, and* $working_temperature_low \in \mathsf{Unknown}(\mathcal{P})$*. Hence* $\{\sim engine_ok, heat\} \subseteq \mathcal{P}^{accepted}$*, and* $engine_ok \in \mathcal{P}^{rejected}$*.*

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

We are now considering logic programs containing default negation not. A *program transformation* is a relation $\mapsto$ between ground logic programs [BD97,BD99,BDFZ01]. A semantics SEM allows a transformation $\mapsto$ iff $\mathrm{SEM}(\mathcal{P}_1) = \mathrm{SEM}(\mathcal{P}_2)$, 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 33 (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.,* $HEAD(\mathcal{P}) = \{H | H \leftarrow \mathcal{B}^+, \mathsf{not}\,\mathcal{B}^- \in \mathcal{P}\}$*. 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* $\mathsf{not}\,B \in \mathcal{B}$ *such that there is no rule about* B *in* $\mathcal{P}_1$*, i.e.,* $B \notin HEAD(\mathcal{P}_1)$*, and* $\mathcal{P}_2 = (\mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\}) \cup \{H \leftarrow (\mathcal{B} \setminus \{\mathsf{not}\,B\})\}$*.*

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* $\mathsf{not}\,B \in \mathcal{B}$ *such that* B *appears as a fact in* $\mathcal{P}_1$*, and* $\mathcal{P}_2 = \mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\}$*.*

SUB (Deletion of non-minimal rules): *Program* $\mathcal{P}_2$ *results from pro-*

gram $\mathcal{P}_1$ 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 $\mathcal{P}_2 = \mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}'\}$.*

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 $\mathcal{P}_2 = \mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\} \cup \{H \leftarrow ((\mathcal{B} \setminus \{B\}) \cup \mathcal{B}') \mid B \leftarrow \mathcal{B}' \in \mathcal{P}_1\}$.*

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 $\mathcal{P}_2 = \mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\}$.*

A program $\mathcal{P}'$ is a normalform *of a program $\mathcal{P}$ wrt. a transformation "$\mapsto$" iff $\mathcal{P} \mapsto_* \mathcal{P}'$, where $\mapsto_*$ denotes the reflexive-transitive closure of $\mapsto$, 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 *weakly terminating* (i.e., each 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 fair order, we eventually arrive at a normalform $norm_{\mathrm{WFS}}(\mathcal{P})$). This normalform $norm_{\mathrm{WFS}}(\mathcal{P})$ 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 34 ($\mathbf{SEM_{min}}$) *We define $SEM_{min}(\mathcal{P}) = \langle \mathcal{P}^{true}, \mathcal{P}^{false}, \mathcal{P}^{undef} \rangle$ for any nlp $\mathcal{P}$, where*

$$\begin{aligned} \mathcal{P}^{true} &= \{H \mid H \leftarrow \in \mathcal{P}\} \\ \mathcal{P}^{false} &= \{H \mid H \in \mathcal{L}_P \setminus HEAD(\mathcal{P})\} \\ \mathcal{P}^{undef} &= \{H \mid H \in \mathcal{L}_P \setminus (\mathcal{P}^{true} \cup \mathcal{P}^{false})\} \end{aligned}$$

To illustrate our transformations, we consider the following example taken from [DOZ01]:

Example 35 (Computing WFS) *We consider the program $\mathcal{P}_1$ and reduce it as follows:*

$$\begin{array}{l} p \\ q \leftarrow \mathsf{not}\, p \\ q \leftarrow t, \mathsf{not}\, p \\ s \leftarrow \mathsf{not}\, q \\ q \leftarrow r \\ r \leftarrow q \end{array} \quad \rightarrow_{\boldsymbol{SUB}} \quad \begin{array}{l} p \\ q \leftarrow \mathsf{not}\, p \\ s \leftarrow \mathsf{not}\, q \\ q \leftarrow r \\ r \leftarrow q \end{array} \quad \rightarrow_{\boldsymbol{RED^-}} \quad \begin{array}{l} p \\ s \leftarrow \mathsf{not}\, q \\ q \leftarrow r \\ r \leftarrow q \end{array}$$

In the next step, we can apply ***UNFOLD*** *to one of the two last rules to get:*

$$\begin{array}{l} p \\ s \leftarrow \mathsf{not}\, q \\ q \leftarrow q \\ r \leftarrow q \end{array}$$

Now we can delete the resulting tautology by the application of ***TAUT*** *and then use* $\boldsymbol{Red}^+$

$$\begin{array}{lll} p & & p \\ s \leftarrow \mathsf{not}\, q & \rightarrow_{\boldsymbol{RED}^+} & s \\ r \leftarrow q & & r \leftarrow q \end{array}$$

Finally applying ***UNFOLD*** *to the last one, we get to* $norm_{WFS}(\mathcal{P}_1)$:

$$\begin{array}{l} p \\ s \end{array}$$

Thus, the wellfounded semantics of $\mathcal{P}_1$ *is:*

$$WFS(\mathcal{P}_1) = \{p, s, \mathsf{not}\, q, \mathsf{not}\, t, \mathsf{not}\, r\}$$

Theorem 36 (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 for *NLP* as the ones described above also hold for a *DeLP* program. Such an analysis is very complicated for the whole class *DeLP*, where we have not only two sorts of rules, *strict* and *defeasible* rules, but also two different kinds of negation, $\sim$ and not. Adapting the transformation rules presented in Section 3 to this class of programs is a nontrivial task. In fact, even defining a semantics for general programs in *DeLP* is highly nontrivial and subject of ongoing research.

In our analysis, we will therefore focus first on $DeLP_{neg}$ (i.e., *DeLP* with strict negation "$\sim$"). As the transformations in [BDFZ01,BD98] are defined with respect to a *NLP* setting, we will adapt them accordingly. Therefore, we extend our previous terminology to be applied to a $DeLP_{neg}$ program $\mathcal{P}$

(thus HEAD($\mathcal{P}$) will stand for all heads of rules in $\mathcal{P}$, etc.), distinguishing strict rules from defeasible rules when needed. In Section 4.2 we will consider $DeLP_{not}$ (i.e., *DeLP* with default negation not). In that case, a similar analysis will be performed.

The following propositions provide ways of determining whether two *dlp* programs have the same semantics. These results will be used in the following sections.

Proposition 37 *Let $\mathcal{P}$ and $\mathcal{P}'$ be two dlp programs. If $Args(\mathcal{P}) = Args(\mathcal{P}')$, then $SEM_{DeLP}(\mathcal{P}') = SEM_{DeLP}(\mathcal{P})$.*

PROOF. This is a direct consequence of the Definition 31, since the semantics of *DeLP* is entirely determined by relationships among arguments.

The converse does not hold, as shown in the following example.

Example 38 *Let $\mathcal{P}_1 = \{ p \prec q,\ p \prec r,\ q \leftarrow ,\ r \leftarrow \}$, and let $\mathcal{P}_2 = \{ p \prec q,\ q \leftarrow ,\ r \leftarrow \}$. Clearly, $SEM_{DeLP}(\mathcal{P}_1) = SEM_{DeLP}(\mathcal{P}_2)$, since $\{p, q, r\} = \mathcal{P}_1^{accepted} = \mathcal{P}_2^{accepted}$. However $Args(\mathcal{P}_1) \neq Args(\mathcal{P}_2)$ (since $\langle \{p \prec r\}, p\rangle$ is an argument in $\mathcal{P}_1$ but not in $\mathcal{P}_2$).*

Definition 39 (Isomorphic dialectical trees) *Given two arguments $\langle \mathcal{A}_1, Q_1\rangle$ and $\langle \mathcal{A}_2, Q_2\rangle$, their associated dialectical trees $\mathcal{T}_{\langle \mathcal{A}_1, Q_1\rangle}$ and $\mathcal{T}_{\langle \mathcal{A}_2, Q_2\rangle}$ will be* isomorphic *iff*

(1) $Q_1 = Q_2$, and both $\langle \mathcal{A}_1, Q_1\rangle$ and $\langle \mathcal{A}_2, Q_2\rangle$ have no defeaters, or
(2) $\mathcal{T}_{\langle \mathcal{A}_1, Q_1\rangle}$ has $\mathcal{T}_1, \ldots, \mathcal{T}_k$ as immediate subtrees, and $\mathcal{T}_{\langle \mathcal{A}_2, Q_2\rangle}$ has $\mathcal{T}_1', \ldots, \mathcal{T}_k'$ as immediate subtrees, and there exists a one-to-one correspondence $f : \{\mathcal{T}_1, \ldots, \mathcal{T}_k\} \mapsto \{\mathcal{T}_1', \ldots, \mathcal{T}_k'\}$, such that
   (a) $\mathcal{T}_i$ and $f(\mathcal{T}_i)$ are isomorphic, $i = 1, \ldots, k$, and
   (b) The root of $\mathcal{T}_i$ is a proper (resp. blocking) defeater for $\langle \mathcal{A}_1, Q_1\rangle$ and the root of $f(\mathcal{T}_i)$ is a proper (resp. blocking) defeater for $\langle \mathcal{A}_2, Q_2\rangle$, for $i = 1, \ldots, k$.

Proposition 40 *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two $DeLP_{not}$ programs, such that $\mathcal{T}_{\langle \mathcal{A}_1, Q_1\rangle}$ is the associated dialectical tree for an argument $\langle \mathcal{A}_1, Q_1\rangle$ in $\mathcal{P}_1$, and $\mathcal{T}_{\langle \mathcal{A}_2, Q_2\rangle}$ is the associated dialectical tree for an argument $\langle \mathcal{A}_2, Q_2\rangle$ in $\mathcal{P}_2$. If $\mathcal{T}_{\langle \mathcal{A}_1, Q_1\rangle}$ and $\mathcal{T}_{\langle \mathcal{A}_2, Q_2\rangle}$ are isomorphic, then $Q_1 \in \mathcal{P}_1^{accepted}$ (resp. $\mathcal{P}_1^{rejected}$, $\mathcal{P}_1^{undef}$) iff $Q_2 \in \mathcal{P}_2^{accepted}$ (resp. $\mathcal{P}_2^{rejected}$, $\mathcal{P}_2^{undef}$).*

PROOF. This proposition is direct consequence of the definition of marking of a dialectical tree (Definition 27).

Corollary 41 *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two DeLP$_{\mathsf{not}}$ programs, such that HEAD($\mathcal{P}_1$) = HEAD($\mathcal{P}_2$). Suppose that for any literal Q in HEAD($\mathcal{P}_1$), there exists a dialectical tree $\mathcal{T}_{\langle \mathcal{A},Q\rangle}$ in $\mathcal{P}_1$ iff there exists an isomorphic dialectical tree $\mathcal{T}_{\langle \mathcal{B},Q\rangle}$ in $\mathcal{P}_2$. Then $SEM_{DeLP}(\mathcal{P}_1) = SEM_{DeLP}(\mathcal{P}_1)$.*

4.1 Transformation Properties in 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 each transformation, $\mathcal{P}_1$ and $\mathcal{P}_2$ denote ground *dlp* programs. Some transformation rules have special requirements which appear in boldface.

RED$^+_{neg}$: Program $\mathcal{P}_2$ will result from program $\mathcal{P}_1$ by **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 $\mathcal{P}_2 = (\mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\}) \cup \{H \leftarrow (\mathcal{B} \setminus \{\sim B\})\}$.

RED$^-_{neg}$: Program $\mathcal{P}_2$ will result from program $\mathcal{P}_1$ by **RED$^-$** (written $\mathcal{P}_1 \mapsto_{Mneg} \mathcal{P}_2$) 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 $\mathcal{P}_2 = \mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\}$.

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

UNFOLD$_{neg}$: Suppose program $\mathcal{P}_1$ contains a **strict** rule $H \leftarrow \mathcal{B}$ 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$_{neg}$** (written $\mathcal{P}_1 \mapsto_{Uneg} \mathcal{P}_2$) iff there is a positive literal $B \in \mathcal{B}$ [2] which **does not appear as head of a defeasible rule in $\mathcal{P}_1$**, such that $\mathcal{P}_2 = \mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\} \cup \{H \leftarrow ((\mathcal{B} \setminus \{B\}) \cup \mathcal{B}') \mid B \leftarrow \mathcal{B}' \in \mathcal{P}_1\}$.

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 $i = 1, \ldots, n$).

TAUT$_{neg}$: Program $\mathcal{P}_2$ will result from program $\mathcal{P}_1$ by **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 $\mathcal{P}_2 = \mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\}$.

First we consider **RED$^+_{neg}$**. This transformation rule *does not hold* for strict negation. Note that whereas **RED$^+$** captures the idea that $\mathsf{not}\, A$ trivially holds whenever A cannot be derived (and for that reason $\mathsf{not}\, A$ can be deleted), the

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

same principle cannot be applied to $\sim A$, which holds whenever *there is a derivation for* $\sim A$.

Example 42 *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 considered* $\mathcal{P}' = \boldsymbol{RED}^{+}_{neg}(\mathcal{P})$ *we would 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 43

Proposition 43 *Let* $\mathcal{P}$ *be a DeLP$_{neg}$ program. Let* $\mathcal{P}'$ *be the resulting program of applying* $\boldsymbol{RED}^{-}_{neg}$, *i.e.,* $\mathcal{P} \mapsto_{Mneg} \mathcal{P}'$. *Then* $SEM_{DeLP}(\mathcal{P}') = SEM_{DeLP}(\mathcal{P})$.

PROOF. Let $\mathcal{P}$ be a *DeLP$_{neg}$* program, and let $(A \leftarrow) \in \mathcal{P}$. Furthermore, 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, each argument that can be built from $\mathcal{P}$ can also be built from $\mathcal{P}' = \mathcal{P} \setminus \{r\}$. 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 45. 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 44

Example 44 *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* $\mathcal{P}' = \mathcal{P} \setminus \{(p \prec q_1, q_2)\}$, *then we get two arguments which block each other (*$\mathcal{A} = \{ (p \prec q_1) \}$ *for p and* $\mathcal{B} = \{ (\sim p \prec q_2) \}$ *for* $\sim p$*).*

Proposition 45 *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* $SEM_{DeLP}(\mathcal{P}) = SEM_{DeLP}(\mathcal{P}')$.

PROOF. Clearly, $\mathcal{P} = \mathcal{P} \setminus \{r \mid r \text{ is a non-minimal rule }\}$. 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 each 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 each literal Q_i $(i = 1, \ldots, j)$ there are arguments $\langle \mathcal{B}_1, Q_1 \rangle$,...,$\langle \mathcal{B}_j, Q_j \rangle$. 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 $\mathcal{A}' \subset \mathcal{A}$ must be an argument for H. This means that $\mathcal{A}$ is not an argument according to Definition 10, because it does not satisfy condition 3. In any case, the rule r can be removed from $\mathcal{P}$, without affecting the arguments that can be obtained from $\mathcal{P}$. Therefore $Args(\mathcal{P}) = Args(\mathcal{P}')$ ($\mathcal{P}' = \mathcal{P} \setminus \{r\}$). Hence $\text{SEM}_{DeLP}(\mathcal{P}) = \text{SEM}_{DeLP}(\mathcal{P}')$.

Let us now consider **UNFOLD**$_{neg}$. As indicated in its definition, this property is only defined for a certain class of strict rules. It does not hold for defeasible rules, as shown in Example 46. It does not hold for strict rules in general either: we imposed the additional condition that no defeasible rule has the same head as the literal which is being removed when applying "unfolding". The reason for doing so is shown in Example 47.

Example 46 (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 to defeasible rules, we get $\mathcal{P}'$ = (Π, Δ'), 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 47 *Let $\mathcal{P} = (\Pi, \Delta)$ be a dlp, where Π = { ($p \leftarrow q, s$), ($q \leftarrow f_1$), ($q \leftarrow f_2$), ($s \leftarrow$) }, and Δ = { $q -\!\prec s$ }. If we could apply* **UNFOLD**$_{neg}$ *on rule ($p \leftarrow q, s$) wrt. the literal q, we would get the program $\mathcal{P}$' = $\mathcal{P} \setminus \{(p \leftarrow q, s)\} \cup \{(p \leftarrow f_1, s), (p \leftarrow f_2, s)\}$. But $\mathcal{A}_1$ = { $q -\!\prec s$ } is an argument for p in $\mathcal{P}$, but it does not exist in $\mathcal{P}$'.*

In order to simplify the analysis of **UNFOLD**$_{neg}$, we will define a special transformation **UNFOLD**$^{r_i}_{neg}$ corresponding to **UNFOLD**$_{neg}$ applied to a particular **UNFOLD**$_{neg}$-related rule r_i.

Definition 48 (Transformation UNFOLD$^{r_i}_{neg}$) *Suppose program $\mathcal{P}_1$ contains a* ***strict*** *rule $H \leftarrow \mathcal{B}$ 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$^{r_i}_{neg}$*** *(written $\mathcal{P}_1 \mapsto^{r_i}_{Uneg} \mathcal{P}_2$) iff there is a positive literal $B \in \mathcal{B}$ which* ***does not appear as head of a defeasible rule in*** *$\mathcal{P}_1$, such that $\mathcal{P}_2 = \mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\} \cup \{H \leftarrow ((\mathcal{B} \setminus \{B\}) \cup \mathcal{B}') \mid r_i = B \leftarrow \mathcal{B}' \in \mathcal{P}_1\}$. (Such r_i are called* ***UNFOLD$_{neg}$*** *related.)*

Proposition 49 *Let $\mathcal{P}_1$ be a DeLP$_{neg}$ program which contains a* ***strict*** *rule $r{=}H \leftarrow \mathcal{B}$, such that r_1, r_2, ..., r_k are all those rules in $\mathcal{P}_1$ that are* ***UNFOLD$_{neg}$***-*related to r. Consider the sequence of programs $\mathcal{P} = \mathcal{P}_1 \mapsto^{r_1}_{Uneg} \mathcal{P}_2 \mapsto^{r_2}_{Uneg}, \ldots, \mathcal{P}_k \mapsto^{r_k}_{Uneg} \mathcal{P}'$. Then $\mathcal{P} \mapsto_{Uneg} \mathcal{P}'$ wrt rule r.*

PROOF. Direct consequence of Definition 48 and the definition of **UNFOLD**$_{neg}$.

We present next a particular property of immediate subarguments in *DeLP*$_{neg}$, which will allow us to show that the transformation $\mapsto^{r_i}_{Uneg}$ preserves semantics when applied to a given *DeLP*$_{neg}$ program.

Proposition 50 *Let $\langle \mathcal{A}, H \rangle$ be an argument in DeLP$_{neg}$, such that the last rule used in the derivation is the strict rule $H \leftarrow P_1, \ldots, P_k$. Then all immediate subarguments $\langle \mathcal{A}_1, P_1 \rangle$ ..., $\langle \mathcal{A}_k, P_k \rangle$ are such that $\mathcal{A}_i = \mathcal{A}$, $\forall i = 1, \ldots, k$.*

PROOF. Since $\langle \mathcal{A}, H \rangle$ is an argument, then $\Pi \cup \mathcal{A} \vdash H$, such that there exists a defeasible derivation $S = [r_1, \ldots, r_k]$ where $r_1 = H \leftarrow P_1, \ldots, P_k$. Clearly, the sequence $S' = [r_2, \ldots, r_k]$ provides a defeasible derivation for every element of the sequence of goals $G{=}[P_1, \ldots, P_k]$, using the same set $\mathcal{A}$ of defeasible information as in S. In particular, $\Pi \cup \mathcal{A} \vdash P_i$, $\forall i = 1, \ldots, k$, such that $\mathcal{A}$ is minimal and non-contradictory. Thus $\mathcal{A}$ is an argument for P_i, $\forall i = 1, \ldots, k$.

Proposition 51 *Let $\mathcal{P}_1$ be a DeLP$_{neg}$ program, and let $\mathcal{P}_2$ be the program resulting from applying $\mapsto^{r_i}_{Uneg}$ wrt some rule r_i.*

Let $\langle \mathcal{A}, H \rangle$ be an argument in $\mathcal{P}_1$ affected by the application of $\mapsto^{r_i}_{Uneg}$. Then $\langle \mathcal{A}, H \rangle$ is also an argument in $\mathcal{P}_2$, and $Args(\mathcal{P}_1) = Args(\mathcal{P}_2)$.

PROOF. Let $\mathcal{P}_1 = (\Pi, \Delta)$ be a *DeLP*$_{neg}$ program. Let $\langle \mathcal{A}, Q \rangle$ be an argument in $\mathcal{P}_1$. We can assume that (i) a strict rule $r = H \leftarrow \mathcal{B}$ is used in the defeasible

derivation of Q from $\Pi \cup \mathcal{A}$, and (ii) r is **UNFOLD**$_{neg}$-related to other rule r_i (otherwise $Args(\mathcal{P}_1) = Args(\mathcal{P}_2)$, and the proposition holds trivially).

Since rule r was applied in the defeasible derivation of Q from $\Pi \cup \mathcal{A}$, there exists an argument $\langle \mathcal{S}, H \rangle$ which is a subargument of $\langle \mathcal{A}, Q \rangle$, such that the last rule used in the defeasible derivation of $\langle \mathcal{S}, H \rangle$ is r. The strict rule r can be written as

$$r = H \leftarrow B, L_1, \ldots, L_k \tag{1}$$

From Proposition 50, we get that $\langle \mathcal{S}, B \rangle$, $\langle \mathcal{S}, L_1 \rangle$, ..., $\langle \mathcal{S}, L_k \rangle$ are immediate subarguments of $\langle \mathcal{S}, H \rangle$.

Consider r_i=$B \leftarrow \mathcal{B}$, which is the last rule used in the defeasible derivation of $\langle \mathcal{S}, B \rangle$, such that r is **UNFOLD**$_{neg}$-related to r_i. Since r_i is a strict rule, it will have the form

$$r_i = B \leftarrow P_1, \ldots, P_m. \tag{2}$$

From Proposition 50, we get that $\langle \mathcal{S}, P_1 \rangle$, $\langle \mathcal{S}, P_2 \rangle$, ..., $\langle \mathcal{S}, P_m \rangle$ are immediate subarguments of $\langle \mathcal{S}, B \rangle$. Thus, argument $\langle \mathcal{S}, H \rangle$ in $\mathcal{P}_1$ is such that $\langle \mathcal{S}, P_1 \rangle$, ..., $\langle \mathcal{S}, P_m \rangle$ and $\langle \mathcal{S}, L_1 \rangle$, ..., $\langle \mathcal{S}, L_k \rangle$ are also arguments in $\mathcal{P}_1$.

Assume we apply $\mapsto^{r_i}_{Uneg}$ to $\mathcal{P}_1$, resulting in a new $DeLP_{neg}$ program $\mathcal{P}_2$. From Definition 48, we have:

$$\mathcal{P}_2 = \mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\} \cup \{H \leftarrow ((\mathcal{B} \setminus \{B\}) \cup \mathcal{B}') \mid r_i = B \leftarrow \mathcal{B}' \in \mathcal{P}_1\}$$

In this case we get $\mathcal{P}_2 = \mathcal{P}_1 \setminus \{r\} \cup \{r'\}$, where r' is the rule

$$r' = H \leftarrow L_1, \ldots, L_k, P_1, \ldots, P_m \tag{3}$$

Clearly, $\langle \mathcal{S}, L_i \rangle$, $i = 1, \ldots, k$ and $\langle \mathcal{S}, P_i \rangle$, $i = 1, \ldots, m$ are also arguments in $\mathcal{P}_2$, and in particular $\langle \mathcal{S}, H \rangle$ is also an argument in $\mathcal{P}_2$. Note that no new argument other than $\langle \mathcal{S}, H \rangle$ is generated in $\mathcal{P}_2$, since the subarguments of $\langle \mathcal{S}, H \rangle$ in $\mathcal{P}_1$ and $\langle \mathcal{S}, H \rangle$ in $\mathcal{P}_2$ are the same. Thus $Args(\mathcal{P}_1) = Args(\mathcal{P}_2)$.

Corollary 52 *Let $\mathcal{P}$ be a $DeLP_{neg}$ program, and let $\mathcal{P}'$ be the program resulting from applying **UNFOLD**$_{neg}$ wrt some rule r in $\mathcal{P}$. Then $SEM_{DeLP}(\mathcal{P}) = SEM_{DeLP}(\mathcal{P}')$.*

PROOF. Follows directly from Proposition 49 by repeated application of $\mapsto^{r_i}_{Uneg}$, for each r_i which is $\mathbf{UNFOLD}_{neg}$-related with r.

Let us now consider tautology elimination.

Proposition 53 *Let $\mathcal{P}$ be a $DeLP_{neg}$ program, and $\mathcal{P}'$ the program resulting from applying $\boldsymbol{TAUT}_{neg}$ to $\mathcal{P}$, i.e., $\mathcal{P} \mapsto_{Tneg} \mathcal{P}'$ Then $SEM_{DeLP}(\mathcal{P}) = SEM_{DeLP}(\mathcal{P}')$.*

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 $\Pi \setminus \{r\} \cup \mathcal{A}$. Thus, there exists a derivation for Q from $\Pi \setminus \{r\} \cup \mathcal{A}$ (the same holds the other way around). Therefore, $\langle \mathcal{A}, Q \rangle \in Args(\mathcal{P}$ iff $\langle \mathcal{A}, Q \rangle \in Args(\mathcal{P} \setminus \{r\})$. 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 $r = P \multimap P, S_1, \ldots, S_k$. 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 $P \multimap P, S_1, \ldots, S_k$ can be used in building an argument. Therefore, $\langle \mathcal{A}, P \rangle \in Args(\mathcal{P})$ iff $\langle \mathcal{A}, P \rangle \in Args(\mathcal{P} \setminus \{r\})$.

It must be remarked that defeasible information in a given argument is represented through the defeasible rules used in its construction. This explains why we have to restrict ourselves to strict rules when considering $\mathbf{SUB}_{neg}$ and $\mathbf{UNFOLD}_{neg}$. Performing such transformations on defeasible rules may cause the loss of specificity information present in the antecedent of those rules (i.e., information that distinguishes a defeasible rule as 'more informed' than another). A similar situation will arise with respect to $\mathbf{SUB}_{not}$ and $\mathbf{UNFOLD}_{not}$, as presented in Section 4.2.

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

$DeLP_{\mathsf{not}}$ is the subclass of programs in $DeLP$ which contain only default negation not, but no strict negation $\sim$. This class can also 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 $\sim P$ can appear. The attack relationship among arguments is defined in terms of default literals: an argument $\langle \mathcal{A}, Q_1 \rangle$ accounts for a *counterargument* for an argument $\langle \mathcal{B}, Q_2 \rangle$ if $\mathsf{not}\, Q_1$ is used as an *assumption* in the defeasible derivation of Q_2 from $\Pi \cup \mathcal{B}$.

Assumption literals are the only possible points for attack in $DeLP_{\mathsf{not}}$. In fact, we now restrict our framework in that we allow in Definition 10 only

assume $\sim A$ where A is an atom. That is, we do not allow assume A literals. Thus the set $\mathcal{H}_{\langle \mathcal{A},Q \rangle}$ denotes in this section the set of assumption literals in $\langle \mathcal{A}, Q \rangle$ where all literals are (strictly) negated atoms. The reason is that we want to have as much assume $\sim A$ as is consistently possible: these negated atoms do represent the closed world assumption which is always implicit in such a setting.

An argument involving an assumption assume $\sim A$ will be attacked by any other argument concluding A. In order to capture this situation, the notion of a *contradictory set of literals* has been extended after Definition 6 to consider assumption literals.

Strict arguments $\langle \emptyset, R \rangle$ have the special property of defeating any other argument involving an assumption literal, as shown in the following proposition.

Proposition 54 *Let $\mathcal{P}$ be a DeLP$_{not}$ program, and let $\langle \mathcal{A}, Q \rangle$ be an argument in $\mathcal{P}$ such that Q follows from $\mathcal{A}$ using* assume $\sim R$ *as an assumption. If $\langle \emptyset, R \rangle$, then $\langle \mathcal{A}, Q \rangle$ is not a justification.*

PROOF. Clearly $\langle \emptyset, R \rangle$ is a counterargument for $\langle \mathcal{A}, Q \rangle$, in particular (according to specificity) a defeater. Since $\langle \emptyset, R \rangle$ has no defeaters (as discussed on page 12), the dialectical tree with root $\langle \mathcal{A}, Q \rangle$ will have a children node $\langle \emptyset, R \rangle$, which will turn out to be marked as U (according to Definitions 27). Hence $\langle \mathcal{A}, Q \rangle$ will be marked as D, so that $\langle \mathcal{A}, Q \rangle$ is not a justification.

The precise semantics for *DeLP$_{not}$* depends on the analogue of Definitions 14 and 18 and the appropriate notion of a dialectical tree. Suitable definitions capture different semantics ([GSC98]). But independently of these notions, it can be stated that not Q will not hold whenever Q can be ultimately defeated. In particular, not Q will not hold whenever there is a strict argument for Q. In this respect, *DeLP$_{not}$* naturally extends the intended meaning of default negation in traditional logic programming (not H holds iff H fails to be finitely proven). This fact also suffices to decide which of the transformation properties are satisfied or to give counterexamples.

Since a *DeLP$_{not}$* program does not involve strict negation, many problems considered in Subsection 4.1 do not arise. New transformations $\mathbf{RED}^{+}_{not}$, $\mathbf{RED}^{-}_{not}$, $\mathbf{SUB}_{not}$, $\mathbf{UNFOLD}^{r}_{not}$, $\mathbf{UNFOLD}_{not}$ and $\mathbf{TAUT}_{not}$ can be defined, with the same meaning as the ones introduced in Subsection 4.1 for *DeLP$_{neg}$*, but referring to *DeLP$_{not}$* programs. Similarly, we will use the $\mathcal{P} \mapsto_{R^{+}not} \mathcal{P}'$ (resp. $\mapsto_{R^{-}not}$, $\mapsto_{Snot}$, $\mapsto_{Unot}$, $\mapsto^{r}_{Unot}$, $\mapsto_{Tnot}$) to denote the *DeLP$_{not}$* program $\mathcal{P}'$ resulting from $\mathcal{P}$ by application of the transformation $\mathbf{RED}^{+}_{not}$ (resp. $\mathbf{RED}^{-}_{not}$, $\mathbf{SUB}_{not}$, $\mathbf{UNFOLD}_{not}$, $\mathbf{UNFOLD}^{r}_{not}$, $\mathbf{TAUT}_{not}$).

For each transformation, we will show that the resulting transformed program is equivalent to the original one. In the case of $\mathbf{SUB}_{not}$ and $\mathbf{UNFOLD}_{not}$, we restrict ourselves to strict rules, since these transformations do not hold when applied to defeasible rules (as shown in Examples 46 and 44).

Proposition 55 *Let $\mathcal{P}$ be a DeLP$_{not}$ program. Let $\mathcal{P}'$ be the DeLP$_{not}$ program resulting from $\mathcal{P} \mapsto_{R^+not} \mathcal{P}'$. Then $SEM_{DeLP}(\mathcal{P}) = SEM_{DeLP}(\mathcal{P}')$.*

PROOF. Let $\mathcal{P}$ be a *DeLP$_{not}$* program, such that $r = P \prec Q_1, \ldots, \mathsf{not}\, Q, \ldots, Q_k$ is a defeasible rule in $\mathcal{P}$, and there is no rule about Q in $\mathcal{P}$. Let $\mathcal{P}'$ be the *DeLP$_{not}$* program resulting from applying $\mapsto_{R^+not}$ to $\mathcal{P}$ on rule r.

Let H be an arbitrary literal in $\mathcal{P}$, such that rule r is used in building the defeasible derivation of some argument $\langle \mathcal{B}, S \rangle$, so that $\mathsf{assume} \sim Q$ is an assumption literal in $\langle \mathcal{B}, S \rangle$. Since $\mathcal{P}' =_{def} \mathcal{P} \setminus \{r\} \cup \{P \prec Q_1, \ldots, Q_k\}$, it is clear that S has also a defeasible derivation from $\mathcal{B} \setminus \{r\} \cup \{P \prec Q_1, \ldots, Q_k\}$, which is minimal and non-contradictory. Hence we have the argument $\langle \mathcal{B} \setminus \{r\} \cup \{P \prec Q_1, \ldots, Q_k\}, S \rangle$ in $\mathcal{P}'$.

Since there is no rule with head Q in $\mathcal{P}$, there exists no argument $\langle \mathcal{C}, Q \rangle$ in $\mathcal{P}$ and hence no counterargument for $\langle \mathcal{B}, S \rangle$ at $\mathsf{assume} \sim Q$. Therefore each defeater for $\langle \mathcal{B}, S \rangle$ in $\mathcal{P}$ is also a defeater for $\langle \mathcal{B}', S \rangle$ in $\mathcal{P}'$, where $\mathcal{B}' = \mathcal{B} \setminus \{r\} \cup \{P \prec Q_1, \ldots, Q_k\}$. The same line of reasoning applies if r is a strict rule $P \leftarrow Q_1, \ldots, Q_k$.

Hence each dialectical tree $\mathcal{T}$ in $\mathcal{P}$ involving $\langle \mathcal{B}, S \rangle$ as a node is isomorphic to $\mathcal{T}'$ in $\mathcal{P}'$ involving $\langle \mathcal{B}', S \rangle$ in $\mathcal{P}'$. From Proposition 40 it follows that $\mathrm{SEM}_{DeLP}(\mathcal{P}) = \mathrm{SEM}_{DeLP}(\mathcal{P}')$.

Proposition 56 *Let $\mathcal{P}$ be a DeLP$_{not}$ program. Let $\mathcal{P}'$ be the DeLP$_{not}$ program resulting from $\mathcal{P} \mapsto_{R^-not} \mathcal{P}'$. Then $SEM_{DeLP}(\mathcal{P}) = SEM_{DeLP}(\mathcal{P}')$.*

PROOF. Let $\mathcal{P} = (\Pi, \Delta)$ be a *DeLP$_{not}$* program. Let $r = P \leftarrow Q_1, \ldots, \mathsf{not}\, Q, \ldots, Q_n$ be a strict rule, and assume $Q \leftarrow \in \mathcal{P}$. Assume r is used in a defeasible derivation for building an argument $\langle \mathcal{A}, H \rangle$. Clearly $\Pi \cup \mathcal{A} \vdash Q$ and $\Pi \cup \mathcal{A} \vdash \mathsf{assume} \sim Q$. But this violates condition 2 in Definition 10 (contradiction). Therefore each argument $\langle \mathcal{A}, H \rangle$ in $\mathcal{P}$ is also an argument in $\mathcal{P} \setminus \{r\}$. Hence $Args(\mathcal{P}) = Args(\mathcal{P}')$, and therefore $\mathrm{SEM}_{DeLP}(\mathcal{P}) = \mathrm{SEM}_{DeLP}(\mathcal{P}')$.

Proposition 57 *Let $\mathcal{P}$ be a DeLP$_{not}$ program. Let $\mathcal{P}'$ be the DeLP$_{not}$ program resulting from $\mathcal{P} \mapsto_{Snot} \mathcal{P}'$. Then $SEM_{DeLP}(\mathcal{P}) = SEM_{DeLP}(\mathcal{P}')$.*

PROOF. Let $\mathcal{P}$ be a $DeLP_{not}$ program, and let $r = P \leftarrow \mathcal{B}_1$ be a non-minimal strict rule in $\mathcal{P}$ (i.e., there exists a rule $r' = P \leftarrow \mathcal{B}_2$ such that $\mathcal{B}_2 \subseteq \mathcal{B}_1$). We consider $\mathcal{B}_1 = \mathcal{B}_1^+ \cup \mathsf{not}\,\mathcal{B}_1^-$, distinguishing the set $\mathcal{B}_1^+$ of positive literals from the set $\mathsf{not}\,\mathcal{B}_1^-$ (literals preceded by not). If $\mathcal{B}_2 \subseteq \mathcal{B}_1$, then two situations are to be considered: either $\mathcal{B}_2^+ \subseteq \mathcal{B}_1^+$, or $\mathcal{B}_2^- \subseteq \mathcal{B}_1^-$.

(1) Suppose $\mathcal{B}_2^+ \subseteq \mathcal{B}_1^+$. Then $Args(\mathcal{P}) = Args(\mathcal{P} \setminus \{r\})$, following the same line of reasoning as in Proposition 45.
(2) Suppose that $\mathcal{B}_2^- \subseteq \mathcal{B}_1^-$, $\mathcal{B}_2^+ = \mathcal{B}_1^+$. Suppose there exists an argument $\langle \mathcal{A}, H \rangle$ such that the strict rule $r = P \leftarrow \mathcal{B}_1$ is used in the defeasible derivation of H. Clearly, there is an assumption literal $\mathsf{assume}\sim Q$ in $\mathcal{A}$ for each $\mathsf{not}\,Q$ in $\mathcal{B}_1^-$. Let $\mathcal{H}_1$ be the set of assumption literals in $\mathcal{A}$. It follows that $\mathcal{A} \setminus \mathcal{H}_2$ also provides a defeasible derivation for H using r' instead, where $\mathcal{H}_2$ is the set of assumption literals in r', such that $\mathcal{H}_2 \subseteq \mathcal{H}_1$. But then the defeasible derivation of H using r violates condition 3 in Definition 10. Therefore no argument using r can be built in $\mathcal{P}$, so that $Args(\mathcal{P}) = Args(\mathcal{P} \setminus \{r\})$.

From this analysis it follows that $\mathcal{P} \mapsto_{Snot} \mathcal{P}'$ is such that $\mathrm{SEM}_{DeLP}(\mathcal{P}) = \mathrm{SEM}_{DeLP}(\mathcal{P}')$.

Proposition 58 *Let $\mathcal{P}$ be a $DeLP_{not}$ program. Let $\mathcal{P}'$ be the $DeLP_{not}$ program resulting from $\mathcal{P} \mapsto_{Tnot} \mathcal{P}'$. Then $SEM_{DeLP}(\mathcal{P}) = SEM_{DeLP}(\mathcal{P}')$.*

PROOF. We will consider only the case in which literals preceded by *not* are present in a rule of the form $r = P \leftarrow P, Q_1, \ldots, Q_k$. Otherwise the proof follows the same line of reasoning as in Proposition 53.

(1) Suppose there exists an argument $\langle \mathcal{A}, H \rangle$ in $\mathcal{P}$ such that $\Pi \cup \mathcal{A} \vdash H$ using a strict rule $r = P \leftarrow P, Q_1, \ldots, \mathsf{not}\,Q, \ldots, Q_k$. Then the occurrence of P in the antecedent of r can also be proven from $\Pi \setminus \{r\} \cup \mathcal{A}'$, where $\mathcal{A}' = \mathcal{A} \setminus \{\mathsf{assume}\sim Q\}$. But then $\langle \mathcal{A}, H \rangle$ is not an argument, since it violates condition 3 in Definition 10. Therefore, no rule $r = P \leftarrow P, Q_1, \ldots, \mathsf{not}\,Q, \ldots, Q_k$ can be used in an argument in $\mathcal{P}$. Hence $Args(\mathcal{P}) = Args(\mathcal{P}')$, with $\mathcal{P}' = \mathcal{P} \setminus \{r\}$ so that $\mathrm{SEM}_{DeLP}(\mathcal{P}) = \mathrm{SEM}_{DeLP}(\mathcal{P}')$.
(2) Suppose there exists an argument $\langle \mathcal{A}, H \rangle$ in $\mathcal{P}$ such that $\Pi \cup \mathcal{A} \vdash H$ using a defeasible rule $r = P \multimap P, Q_1, \ldots, \mathsf{not}\,Q, \ldots, Q_k$. The same line of reasoning as above applies, with $\mathcal{A}' = \mathcal{A} \setminus \{r, \mathsf{assume}\sim Q\}$. Therefore $\mathrm{SEM}_{DeLP}(\mathcal{P}) = \mathrm{SEM}_{DeLP}(\mathcal{P}')$.

We present next a property of immediate subarguments in $DeLP_{not}$, similar to the one shown in Proposition 50. Then we will show that the transformation $\mapsto_{Unot}^{r_i}$ preserves semantics when applied to a given $DeLP_{not}$ program.

Proposition 59 *Let* $\langle \mathcal{A}, H \rangle$ *be an argument in DeLP*$_{not}$*, such that the last rule used in the derivation is the strict rule* $H \leftarrow P_1, \ldots, P_k, \mathsf{not}\, L_1, \ldots, \mathsf{not}\, L_j$*, distinguishing literals from assumption literals. Then all immediate subarguments* $\langle \mathcal{A}_1, P_1 \rangle \ldots, \langle \mathcal{A}_k, P_k \rangle$ *are such that* $\mathcal{A}_i = \mathcal{A} \setminus \bigcup_{i=1}^{j} \{\mathsf{assume} \sim L_i\}$, $\forall i = 1, \ldots, k$.

PROOF. Follows from the same line of reasoning in Proposition 50 when considering strict rules with assumption literals.

Proposition 60 *Let* $\mathcal{P}$ *be a DeLP*$_{not}$ *program. Let* $\mathcal{P}'$ *be the DeLP*$_{not}$ *program resulting from* $\mathcal{P} \mapsto^{r_i}_{Unot} \mathcal{P}'$ *wrt a strict rule* r *in* $\mathcal{P}$*. Then* $SEM_{DeLP}(\mathcal{P}) = SEM_{DeLP}(\mathcal{P}')$.

PROOF. Let $\mathcal{P}_1 = (\Pi, \Delta)$ be a *DeLP*$_{not}$ program, and let $\langle \mathcal{A}, Q \rangle$ be an argument in $\mathcal{P}_1$, such that (i) a strict rule $r = H \leftarrow \mathcal{B}$ is used in the defeasible derivation of Q from $\Pi \cup \mathcal{A}$, and (ii) r is **UNFOLD**$_{not}$-related to other rule r_i. If this is not the case, then clearly $Args(\mathcal{P}_1) = Args(\mathcal{P}_2)$, and the proposition holds trivially. We can also assume that $\emptyset \subset \mathcal{B}^- \subseteq \mathcal{B}$, i.e., there is at least one literal preceded by *not* in $\mathcal{B}$; otherwise the proposition follows directly from Proposition 51.

Since rule r was applied in the defeasible derivation of Q from $\Pi \cup \mathcal{A}$, there exists an argument $\langle \mathcal{S}, H \rangle$ which is a subargument of $\langle \mathcal{A}, Q \rangle$, such that the last rule used in the defeasible derivation of $\langle \mathcal{S}, H \rangle$ is r.

The strict rule r can be written as

$$r = H \leftarrow B, L_1, \ldots, L_k, \mathsf{not}\, M_1, \ldots, \mathsf{not}\, M_j \tag{4}$$

distinguishing positive literals from literals preceded by *not*. Let $\mathcal{S}_1 = \mathcal{S} \setminus \bigcup_{i=1}^{j} \{\mathsf{assume} \sim M_i\}$. From Proposition 59, we get that $\langle \mathcal{S}_1, B \rangle$, $\langle \mathcal{S}_1, L_1 \rangle$, $\ldots$, $\langle \mathcal{S}_1, L_k \rangle$ are immediate subarguments of $\langle \mathcal{S}, H \rangle$. Hence we get that

$$\mathcal{H}_{\langle \mathcal{S}, H \rangle} = \mathcal{H}_{\langle \mathcal{S}_1, B \rangle} \cup \bigcup_{i=1}^{k} \mathcal{H}_{\langle \mathcal{S}_1, L_i \rangle} \cup \bigcup_{i=1}^{j} \{\mathsf{assume} \sim M_i\} \tag{5}$$

Consider $r_i{=}B \leftarrow \mathcal{B}$, which is the last rule used in the defeasible derivation of $\langle \mathcal{S}_1, B \rangle$, such that r is **UNFOLD**$_{not}$-related to r_i. Since r_i is an arbitrary strict rule, it will have the form

$$r_i = B \leftarrow P_1, \ldots, P_m, \mathsf{not}\, R_1, \ldots, \mathsf{not}\, R_p. \tag{6}$$

Let $\mathcal{S}_2 = \mathcal{S}_1 \setminus \bigcup_{i=1}^{p} \{\mathsf{assume} \sim R_i\}$ It follows that

$$\mathcal{H}_{\langle \mathcal{S}_1, B\rangle} = \bigcup_{i=1}^{m} \mathcal{H}_{\langle \mathcal{S}_2, P_i\rangle} \cup \bigcup_{j=1}^{p} \{\mathsf{assume} \sim R_j\} \tag{7}$$

Replacing (7) in (5), we get

$$\mathcal{H}_{\langle \mathcal{S}, H\rangle} = \bigcup_{i=1}^{m} \mathcal{H}_{\langle \mathcal{S}_2, P_i\rangle} \cup \bigcup_{j=1}^{p} \{\mathsf{assume} \sim R_j\} \cup \bigcup_{i=1}^{k} \mathcal{H}_{\langle \mathcal{S}_1, L_i\rangle} \cup \bigcup_{i=1}^{j} \{\mathsf{assume} \sim M_i\} \tag{8}$$

Thus, argument $\langle \mathcal{S}, H\rangle$ in $\mathcal{P}_1$ is such that $\mathcal{S} = \mathcal{R}_S \cup \mathcal{H}_{\langle \mathcal{S}, H\rangle}$, where $\mathcal{H}_{\langle \mathcal{S}, H\rangle}$ is defined as in (8). Assume we apply $\mapsto_{Unot}$ to $\mathcal{P}_1$, where the rule r is $\mathbf{UNFOLD}_{not}$-related to r_i, resulting in a new $DeLP_{\mathsf{not}}$ program $\mathcal{P}_2$. From the definition of $\mathbf{UNFOLD}_{not}^{r_i}$, we have:

$$\mathcal{P}_2 = \mathcal{P}_1 \setminus \{H \leftarrow \mathcal{B}\} \cup \{H \leftarrow ((\mathcal{B} \setminus \{B\}) \cup \mathcal{B}') \mid r_i = B \leftarrow \mathcal{B}' \in \mathcal{P}_1\}$$

Consider the original rule r in (4), and the $\mathbf{UNFOLD}_{not}$-related rule r_i in (6). Let $\mathcal{P}_2$ be the $DeLP_{\mathsf{not}}$ program resulting from applying the UNFOLD transformation to r with respect to r_i. In this case we get

$$H \leftarrow \{B, L_1, \ldots, L_k, \mathsf{not}\, M_1, \ldots, \mathsf{not}\, M_j\} \setminus \{B\} \cup \{P_1, \ldots, P_m, \mathsf{not}\, R_1, \ldots, \mathsf{not}\, R_p\}$$

or equivalently

$$H \leftarrow L_1, \ldots, L_k, P_1, \ldots, P_m, \mathsf{not}\, M_1, \ldots, \mathsf{not}\, M_j, \mathsf{not}\, R_1, \ldots, \mathsf{not}\, R_p \tag{9}$$

Let $\mathcal{S}' = \mathcal{S} \setminus \{\bigcup_{i=1}^{j} \{\mathsf{assume} \sim M_i\} \cup \bigcup_{i=1}^{p} \{\mathsf{assume} \sim R_i\}\}$. From Proposition 59, it follows that $\langle \mathcal{S}', L_i\rangle$, $i = 1, \ldots, k$ and $\langle \mathcal{S}', P_i\rangle$, $i = 1, \ldots, m$ are arguments in $\mathcal{P}_2$. In particular, we have

$$\mathcal{H}_{\langle \mathcal{S}', H\rangle} = \bigcup_{i=1}^{k} \mathcal{H}_{\langle \mathcal{S}', L_i\rangle} \cup \bigcup_{i=1}^{m} \mathcal{H}_{\langle \mathcal{S}', P_i\rangle} \cup \bigcup_{j=1}^{p} \{\mathsf{assume} \sim R_i\} \bigcup_{i=1}^{j} \{\mathsf{assume} \sim M_i\} \tag{10}$$

Hence $\mathcal{R}_S \cup \mathcal{H}_{\langle \mathcal{S}', H\rangle}$ is an argument for H in $\mathcal{P}_2$, since every defeasible rule in $\mathcal{P}_1$ is also a defeasible rule in $\mathcal{P}_2$. But from (8) and (10) it follows that $\mathcal{H}_{\langle \mathcal{S}', H\rangle}$

	NLP under wfs	$DeLP_{neg}$	$DeLP_{\mathsf{not}}$
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. 3. Behavior of *NLP*, $DeLP_{neg}$ and $DeLP_{\mathsf{not}}$ under different transformations

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

$= \mathcal{H}_{\langle \mathcal{S},H \rangle}$, and the set $\mathcal{S}' = \mathcal{S}$. Hence, $\langle \mathcal{S}, H \rangle$ is an argument in both $\mathcal{P}_1$ and $\mathcal{P}_2$.

Therefore, we can conclude that for any argument $\langle \mathcal{S}, H \rangle$ in $\mathcal{P}_1$ such that one of the strict rules r used in its defeasible derivation is **UNFOLD**$_{not}$-related to another rule r_i, it follows that $\langle \mathcal{S}, H \rangle$ is also an argument in $\mathcal{P}_2$. Note that no new argument other than $\langle \mathcal{S}, H \rangle$ is generated in $\mathcal{P}_2$, since the subarguments of $\langle \mathcal{S}, H \rangle$ in $\mathcal{P}_1$ and $\langle \mathcal{S}, H \rangle$ in $\mathcal{P}_2$ are the same. Hence $Args(\mathcal{P}_1) = Args(\mathcal{P}_2)$, and therefore $\mathrm{SEM}_{DeLP}(\mathcal{P}_1) = \mathrm{SEM}_{DeLP}(\mathcal{P}_2)$.

Corollary 61 *Let $\mathcal{P}$ be a $DeLP_{\mathsf{not}}$ program. Let $\mathcal{P}'$ be the $DeLP_{\mathsf{not}}$ program resulting from $\mathcal{P} \mapsto_{Unot} \mathcal{P}'$ wrt a strict rule r in $\mathcal{P}$. Then $SEM_{DeLP}(\mathcal{P}) = SEM_{DeLP}(\mathcal{P}')$.*

PROOF. Follows directly from Proposition 49 by repeated application of $\mapsto^{r_i}_{Unot}$, for each r_i which is **UNFOLD**$_{not}$-related with r.

4.3 Relating NLP and $DeLP_{\mathsf{not}}$ under WFS

A natural question is how well-founded semantics WFS relates to $DeLP_{\mathsf{not}}$. The answer is very simple because of our results that the transformation properties are semantics preserving and the fact that programs in normalform have an obvious semantics.

Theorem 62 ($DeLP_{\mathsf{not}}$ extends WFS) *Let $\mathcal{P}$ be a program in NLP. We can look at $\mathcal{P}$ also as a theory in $DeLP_{\mathsf{not}}$. Then all atoms A and default atoms* not *A that are true in $WFS(P)$ are also contained in $SEM_{DeLPnot}(P)$.*

PROOF. As all the transformation properties hold, we can transform $\mathcal{P}$ into a normalform where all rules only have negative body literals (or are empty):

- The atoms true in WFS(P) are, by Theorem 36, exactly those A where there is a rule of the form "$A \leftarrow$ ". But those atoms are certainly justified in $\mathrm{SEM}_{DeLPnot}(P)$.
- All default literals $\mathsf{not}\, A$ that are true in WFS(P) are, by Theorem 36, exactly those A where there is no rule with head A. But then $\mathsf{assume} \sim A \leftarrow$ can be assumed as it can not lead to any contradiction.

Example 63 *Consider the normal logic programs*

$$\mathcal{P}_1 = \{\ (a \leftarrow b),\ (b \leftarrow a),\ (c \leftarrow \mathsf{not}\, a, \mathsf{not}\, b)\ \}$$

$$\mathcal{P}_2 = \{\ (a \leftarrow \mathsf{not}\, b),\ (b \leftarrow a)\ \}$$

$$\mathcal{P}_3 = \{\ (a \leftarrow \mathsf{not}\, b),\ (b \leftarrow \mathsf{not}\, a),\ (c \leftarrow a),\ (c \leftarrow b)\ \}$$

$$\mathcal{P}_4 = \{\ (a \leftarrow b, \mathsf{not}\, d),\ (b \leftarrow a, \mathsf{not}\, d),\ (d \leftarrow \mathsf{not}\, d)\ ,\ (c \leftarrow \mathsf{not}\, a, \mathsf{not}\, b)\ \}$$

$$\mathcal{P}_5 = \{\ (a \leftarrow \mathsf{not}\, b),\ (b \leftarrow \mathsf{not}\, a),\ (a \leftarrow \mathsf{not}\, a)\ \}$$

We analyze the above NLP programs as $DeLP_{\mathsf{not}}$ programs.

- *WFS in $\mathcal{P}_1$ is $\{\mathsf{not}\, a, \mathsf{not}\, b, c\}$. The only argument that can be constructed from $\mathcal{P}_1$ as a $DeLP_{\mathsf{not}}$ program is the one which justifies c.*
  Without the last rule $(c \leftarrow \mathsf{not}\, a, \mathsf{not}\, b)$ no arguments for positive atoms can be constructed.
- *WFS in $\mathcal{P}_2$ is empty. Under $DeLP_{\mathsf{not}}$, no argument can be built, since the only possible set $\{\ \mathsf{assume} \sim b\ \}$ leads to contradiction.*
- *WFS in $\mathcal{P}_3$ is empty. In $DeLP_{\mathsf{not}}$, two sets of assumptions are possible for building arguments: $\mathcal{A}_1 = \{\mathsf{assume} \sim a\ \}$ and $\mathcal{A}_2 = \{\mathsf{assume} \sim b\ \}$. We can build the arguments $\langle \mathcal{A}_1, b\rangle$, $\langle \mathcal{A}_2, a\rangle$, $\langle \mathcal{A}_1, c\rangle$, $\langle \mathcal{A}_2, c\rangle$. Any one of these arguments has a blocking defeater. From Definition 28 it follows that no argument is justified.*
- *WFS in $\mathcal{P}_4$ is $\{\mathsf{not}\, a, \mathsf{not}\, b, c\}$. The only argument that can be constructed from $\mathcal{P}4$ as a $DeLP_{\mathsf{not}}$ program is the one which justifies c.*
  However, without the last rule $c \leftarrow \mathsf{not}\, a, \mathsf{not}\, b$ no argument can be built in $\mathcal{P}_4$ under $DeLP_{\mathsf{not}}$ (there is no defeasible sequence for a nor for b).
- *WFS in $\mathcal{P}_5$ is empty. But in $SEM_{DeLPnot}$ the argument $\{\ \mathsf{assume} \sim b\ \}$ is a justification for a. This is because $\langle \{\mathsf{assume} \sim b\}, a\rangle$ cannot be defeated (the only way to do this would be to find an argument involving the assumption $\mathsf{not}\, a$, but this would lead to a contradiction).*

The last program $\mathcal{P}_5$ shows that $\mathrm{SEM}_{DeLPnot}$ is strictly stronger than WFS.

4.4 Relating NLP and DeLP: Summary

Figure 3 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 $\mathbf{RED}^-$). It is worth noticing that $\mathbf{RED}^-$ 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. $\mathbf{RED}^+$) 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 Related Work and Conclusion

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 has been 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 $\mathsf{not}\, L$ should hold too (similarly, if L then $\mathsf{not} \sim L$). In *DeLP* this "coherence principle" also holds [GSC98].

Finally, it must be remarked the original Simari-Loui formulation [SL92] contains a fixed-point definition that characterizes all justified beliefs. A similar approach was used later by Prakken and 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.

5.2 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 Subsection 4.4. 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 $\mathbf{RED}^{-}$ 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 80s. 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 think 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.

[BD97] Stefan Brass and Jürgen Dix. Characterizations of the Disjunctive Stable Semantics by Partial Evaluation. *Journal of Logic Programming*, 32(3):207–228, 1997.

[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.

[BD02] 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., 2002. 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*, 1(5):497–538, 2001.

[CDSS00] Carlos Chesñevar, Jürgen Dix, Guillermo Simari, and Frieder Stolzenburg. Relating defeasible and normal logic programming through transformation properties. In *Proceedings of the 6th Argentinean Congress on Computer Science, CACIC'2000*, pages 371–381. JAIIO, Buenos Aires, Argentina, October 2000.

[CML00] C. I. Chesñevar, A. Maguitman, and R. P. Loui. Logical models of arguments. *ACM Computing Surveys*, 32(4), 2000. To appear.

[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.

[DFN01] Jürgen Dix, Ulrich Furbach, and Ilkka Niemelä. Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations. In Andrei Voronkov and Alan Robinson, editors, *Handbook of Automated Reasoning, Volume 2, Chapter 18*, pages 1121–1234. Elsevier-Science-Press, 2001.

[DOZ01] Jürgen Dix, Mauricio Osorio, and Claudia Zepeda. A General Theory of Confluent Rewriting Systems for Logic Programming and its Applications. *Annals of Pure and Applied Logic*, 108(1-3):153–188, 2001.

[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.

[DSSF99] Jürgen Dix, Frieder Stolzenburg, Guillermo R. Simari, and Pablo R. Fillottrani. Automating defeasible reasoning with logic programming (DeReLoP). In Stefan Jähnichen and Irene Loiseau, editors, *Proceedings of the 2nd German-Argentinian Workshop on Information Technology*, pages 39–46, Königswinter, 1999.

[Dun93] Phan M. Dung. On the Acceptability of Arguments and its Fundamental Role in Nomonotonic Reasoning and Logic Programming. In *Proc. of the 13th. International Joint Conference in Artificial Intelligence (IJCAI), Chambéry, France*, 1993.

[Gar97] Alejandro J. García. Defeasible logic programming: Definition and implementation. Master's thesis, Dep. de Ciencias de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina, July 1997.

[GL88] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[GL90] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David Warren and Peter Szeredi, editors, *Proceedings of the International Conference on Logic Programming*, pages 579–597. MIT Press, 1990.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–387, 1991. (Extended abstract appeared in: Logic Programs with Classical Negation. *Proceedings of the 7-th International Logic Programming Conference, Jerusalem*, pages 579-597, 1990. MIT Press.).

[GSC98] Alejandro J. García, Guillermo R. Simari, and Carlos I. Chesñevar. An argumentative framework for reasoning with inconsistent and incomplete information. In *Workshop on Practical Reasoning and Rationality*, pages 13–20. 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. *Circumscription*, pages 297–353. Clarendon, Oxford, 1994.

[Llo87] John W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, Heidelberg, New York, 1987.

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

[PV01] Henry Prakken and Gerhard Vreeswijk. Logics for Defeasible Argumentation. In Dov Gabbay, editor, *Handbook of Philosophical Logic.* Kluwer Academic Publisher, 2001. To appear.

[Res77] Nicholas Rescher. *Dialectics, a Controversy-Oriented Approach to the Theory of Knowledge.* State University of New York Press, Albany, USA, 1977.

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

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

[VGRS88] Allen Van Gelder, Kenneth A. Ross, and J. S. Schlipf. Unfounded Sets and well-founded Semantics for general logic Programs. In *Proceedings 7th Symposion on Principles of Database Systems*, pages 221–230, 1988.

[Vre93] Gerard A.W. Vreeswijk. *Studies in Defeasible Argumentation.* PhD thesis, Vrije University, Amsterdam (Holland), 1993.

[vRS91] A. van Gelder, K.A. Ross, and J.S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.