

UNIVERSITÄT
KOBLENZ · LANDAU



Relating Defeasible and Normal Logic Programming through Transformation Properties

Carlos I. Chesñevar, Jürgen Dix,
Frieder Stolzenburg, Guillermo R. Simari

3/2001



Fachberichte
INFORMATIK

Universität Koblenz-Landau
Institut für Informatik, Rheinau 1, D-56075 Koblenz

E-mail: researchreports@infko.uni-koblenz.de,

WWW: <http://www.uni-koblenz.de/fb4/>

Relating Defeasible and Normal Logic Programming through Transformation Properties*

Carlos Iván Chesñevar, `cic@cs.uns.edu.ar`¹

Jürgen Dix, `dix@uni-koblenz.de`²

Frieder Stolzenburg, `stolzen@uni-koblenz.de`²

Guillermo Ricardo Simari, `grs@cs.uns.edu.ar`¹

¹Univ. Nacional del Sur, Av. Alem 1253, (8000) Bahía Blanca, ARGENTINA

²Universität Koblenz-Landau, Rheinau 1, 56075 Koblenz, GERMANY

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 ($\text{not } A$) 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 and the stable semantics Stable.

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.

*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].

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 $A \text{---} \langle B$, 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 A for Q (written $\langle A, Q \rangle$ 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 A , which are better than 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, BD01, 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 WFS. The application of these rules leads to a new, simplified *NLP* P' from which its WFS can be easily read off. In this paper we will focus on finding similar transformation rules for *DeLP*, which can be used to simplify the knowledge encoded in a *DeLP* program. In our analysis, we show that in *DeLP* a complete simplification of the original program cannot be achieved. However, our results suggest some connections between the semantics of classical approaches and logic programming with *DeLP*.

The paper is structured as follows: Section 2 introduces preliminary notions concerning *NLP* and *DeLP*. Section 3 introduces transformations for *NLP*. Section 4 shows how to adapt these transformations for *DeLP*, analyzing 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 [Lif94], program rules

with variables can be viewed as “schemata” that represent their ground instances. However, for an efficient implementation it is not suitable to actually ground a given program. It is much better to leave variables as they are and compute appropriate substitutions (variable bindings). 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 2.1 (Normal logic program P) A normal logic program (*nlp*) P is a finite set of normal program rules. A normal program rule has the form $A \leftarrow L_1, \dots, L_k$, where A is an atom and each L_i is an atom B or its negation $\text{not} B$. If $B = \{L_1, \dots, L_k\}$ is the body of a rule $A \leftarrow B$, we also use the notation $A \leftarrow B^+, \text{not} B^-$, where B^+ (resp. B^-) contains all the positive (resp. negative) body atoms in B .

In NLP, atoms A and negated atoms $\text{not} A$ 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 $\text{not} A$. If $B^+ = B^- = \emptyset$, we say that the rule is a fact and denote it by $A \leftarrow$ (or just by A).

We will now introduce some concepts useful for describing what a semantics of a *nlp* is. Let Prog_L be the set of all normal propositional programs with atoms from a signature L . By L_P we understand the signature of P , i.e. the set of atoms that occur in P . A (partial) interpretation based on a signature L is a disjoint pair of sets $\langle I_1, I_2 \rangle$ such that $I_1 \cup I_2 \subseteq L$. A partial interpretation is total if $I_1 \cup I_2 = L$. We may also view an interpretation $\langle I_1, I_2 \rangle$ as the set of atoms and negated atoms $I_1 \cup \text{not} I_2$.

Definition 2.2 (Semantics SEM) A semantics SEM is a mapping which assigns to each logic program P a set $\text{SEM}(P)$ of (partial) models of P , such that SEM is “instantiation invariant”, i.e. $\text{SEM}(P) = \text{SEM}(\text{ground}(P))$, where $\text{ground}(P)$ denotes the Herbrand instantiation of P . A semantics SEM is called 3-value based if for each program P the partial interpretation $\text{SEM}(P)$ is a 3-valued model¹ of P .

In Section 3 we will consider a particular 3-valued semantics for *nlp* called WFS, which can be computed by applying *transformation rules* on a *nlp* P .

2.2 Defeasible Logic Programs (DeLP)

The DeLP language [SL92, Gar97, GSC98] is defined in terms of two disjoint sets of rules: a set of *strict rules* for representing strict (sound) knowledge, and a set of *defeasible rules* for representing tentative information. Rules will be defined using *literals*. A literal L is an atom p or a negated atom $\sim p$, where the symbol “ \sim ” represents *strong negation*. In addition, we will consider *default negation* with “not” here. We define formally:

¹We equip \leftarrow with the Kleene interpretation, where $\text{undef} \leftarrow \text{undef}$ is considered to be true.

Definition 2.3 (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 “not A ”, where A is a literal.

Definition 2.4 (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, i.e. an atom p or a negated atom $\sim p$ (strict negation), 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 $\{L_1, \dots, L_k\}$ can also be written as $L_0 \leftarrow L_1, \dots, L_k$. 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 2.5 (Defeasible rules $Head \multimap Body$) A defeasible rule is an ordered pair, conveniently denoted as $Head \multimap Body$, the first member of which, $Head$, is a literal, i.e. an atom p or a negated atom $\sim p$ (strict negation), and the second member, $Body$, is a finite set of literals, which may be (additionally) negated with “not” (default negation). A defeasible rule with the head L_0 and body $\{L_1, \dots, L_k\}$ can also be written as $L_0 \multimap L_1, \dots, L_k$, $k > 0$.

Syntactically, the symbol “ \multimap ” 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, \dots). The letter r (possibly indexed) will be used for denoting rule names. Literals (i.e. an atom or a negated atom) will be denoted with capital letters (A, B, \dots), possibly indexed. Sets will be denoted as A, B, \dots , possibly indexed. Logic programs will be usually denoted as P_1, P_2 , etc.

Definition 2.6 (Defeasible logic program $P = (\Pi, \Delta)$) A defeasible logic program (dlp) is a finite set of strict and defeasible rules. If P is a dlp, we will distinguish in P the subset Π of strict rules, and the subset Δ of defeasible rules. When required, we will denote 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. $DeLP_{not}$). 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 2.1). We will associate with every program P a set of *assumable facts* of the form $assume L$, for every literal L in P . Those literals will be given a special meaning in the argumentation framework and they will be used to define the semantics of default negation.

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

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* 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 2.7 (Derivation sequence) *A defeasible derivation for a literal Q in a general dlp P (possibly containing assumable facts) is a finite sequence of (instantiations of) rules in 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 $G_i = [Q_1, \dots, Q_m, \dots, Q_n]$ with $1 \leq m \leq n$.
 - If there is a strict or defeasible rule in P with head L_0 and body $\{L_1, \dots, L_k\}$ such that L_0 and Q_m have the most general unifier σ , then $G_{i+1} = [Q_1, \dots, Q_{m-1}, L_1, \dots, L_k, Q_{m+1}, \dots, Q_n]\sigma$ and $r_{i+1} = (L_0 \leftarrow L_1, \dots, L_k)\sigma$ or $r_{i+1} = (L_0 \prec L_1, \dots, L_k)\sigma$, respectively.
 - If Q_m has the form $\text{not}L$ for some literal L (possibly negated with \sim) and the assumable fact $r = \text{assume}\bar{L}$ is in P , then $G_{i+1} = [Q_1, \dots, Q_{m-1}, Q_{m+1}, \dots, Q_n]$ and $r_{i+1} = r$.

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

Definition 2.8 (Contradictory set of rules) *A set of rules S is contradictory iff there is a defeasible derivation from S for some literal P and its complement \bar{P} , i.e., $S \vdash P$ and $S \vdash \bar{P}$.*

Given a *dlp* 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 2.9 *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).

$pump_fuel_ok \multimap sw1$
 (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).

Figure 1: Set Δ (Example 2.9)

Next we introduce the definition of argument in *DeLP*. Basically, an argument for a literal Q is a defeasible derivation $S = [r_1, \dots, r_k]$ 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 2.10 (Argument) *Given a dlp $P = (\Pi, \Delta)$, we will define $HB_{\text{ass}} = \{\text{assume } L \mid L \text{ is a literal in } P\}$. An argument A for a query Q , denoted $\langle A, Q \rangle$, is defined as $R_A \cup H_A$, where R_A is a subset of ground instances of the defeasible rules of P and $H_A \subseteq HB_{\text{ass}}$, such that:*

1. *there exists a defeasible derivation for Q from $\Pi \cup A$.*
2. *$\Pi \cup A$ is non-contradictory, and*
3. *A is minimal with respect to set inclusion.*

An argument $\langle A, Q \rangle$ is *strict* iff $A = \emptyset$. An argument $\langle A_1, Q_1 \rangle$ is a *sub-argument* of another argument $\langle A_2, Q_2 \rangle$, if $A_1 \subseteq A_2$. Given an argument $\langle A, Q \rangle$, we will also write $H_{\langle A, Q \rangle}$ to denote the set of assumption literals in $\langle A, Q \rangle$. Next we introduce the auxiliary notion of *immediate subargument*, which will be used later in the proofs of Propositions 4.12 and 4.24.

Definition 2.11 (Immediate subarguments) *Let $\langle A, H \rangle$ be an argument, such that $H \leftarrow P_1, \dots, P_k$ is the last strict rule used in the defeasible derivation of H from $\Pi \cup A$. Clearly, in such a case there exist subsets A_1, \dots, A_k of A , which are arguments for P_1, \dots, P_k . We will call $\langle A_1, P_1 \rangle, \dots, \langle A_k, P_k \rangle$ immediate subarguments of $\langle A, H \rangle$.*

Example 2.12 *Consider the dlp program (Π, Δ) , with*

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

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

Example 2.13 *Consider Example 2.9. Then the set*

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

is an argument for engine_ok. The set

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

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

Given a dlp program P , we will denote by $\text{Args}(P)$ the set of all possible arguments $\langle A, Q \rangle$ that can be built from P wrt. arbitrary queries Q . We emphasize that this set consists of pairs $\langle A, Q \rangle$ and not just of arguments A alone. This makes the condition $\text{Args}(P) = \text{Args}(P)'$ much stronger and is important for our Proposition 4.1 to hold.

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

Definition 2.14 (Counterargument) *An argument $\langle A_1, Q_1 \rangle$ counterargues an argument $\langle A_2, Q_2 \rangle$ at a literal Q iff there is a subargument $\langle A, Q \rangle$ of $\langle A_2, Q_2 \rangle$ such that $\Pi \cup \{Q_1, Q\}$ is contradictory.*

Example 2.15 Consider Example 2.13. Then $\langle B, \sim \text{fuel_ok} \rangle$ is a counterargument for $\langle A, \text{engine_ok} \rangle$, since there is a subargument $A' = \{ \text{fuel_ok} \leftarrow \text{pump_fuel_ok}, \text{oil_ok} \leftarrow \text{pump_oil_ok}, \text{engine_ok} \leftarrow \text{fuel_ok}, \text{oil_ok} \}$ for fuel_ok , such that $\Pi \cup \{ \text{fuel_ok}, \sim \text{fuel_ok} \}$ 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 A is a non-defeated argument, counterarguments that could be *defeaters* for A are considered, i.e., counterarguments that are preferred to 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 2.16 (Specificity) Given a dlp program P , let Π_G denote the maximal set of Π that does not contain facts, and let F denote the set of all possible literals that have a defeasible derivation in P .

An argument $\langle A_1, Q_1 \rangle$ is strictly more specific than an argument $\langle A_2, Q_2 \rangle$ (denoted $\langle A_1, Q_1 \rangle \succ \langle A_2, Q_2 \rangle$) if and only if:

1. For all $H \subseteq F$: if $\Pi_G \cup H \cup A_1 \vdash Q_1$ and $\Pi_G \cup H \not\vdash Q_1$, then $\Pi_G \cup H \cup A_2 \vdash Q_2$.
2. There exists $H' \subseteq F$ such that $\Pi_G \cup H' \cup A_2 \vdash Q_2$ and $\Pi_G \cup H' \not\vdash Q_2$ and $\Pi_G \cup H' \cup A_1 \not\vdash Q_1$.

Example 2.17 Consider the following dlp P :

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

Then the set of all literals derivable in P is $F = \{ p, \sim p, f_1, f_2 \}$. Consider the arguments $\langle A_1, p \rangle$ and $\langle A_2, \sim p \rangle$, with $A_1 = \{ p \leftarrow f_1, f_2 \}$ and $A_2 = \{ \sim p \leftarrow f_1 \}$. For every $H \subseteq F$, condition 1 in Definition 2.16 holds. For $H' = \{ f_1 \}$, condition 2 in Definition 2.16 holds. Hence $\langle A_1, p \rangle$ is strictly more specific than $\langle A_2, \sim p \rangle$.

Definition 2.18 (Proper defeater, blocking defeater) An argument $\langle A_1, Q_1 \rangle$ defeats $\langle A_2, Q_2 \rangle$ at a literal Q iff there exists a subargument $\langle A, Q \rangle$ of $\langle A_2, Q_2 \rangle$ such that $\langle A_1, Q_1 \rangle$ counterargues $\langle A, Q \rangle$ at Q , and either:

- (a) $\langle A_1, Q_1 \rangle$ is strictly more specific than $\langle A, Q \rangle$. In this case $\langle A_1, Q_1 \rangle$ is called a proper defeater of $\langle A, Q \rangle$.
- (b) Neither $\langle A_1, Q_1 \rangle$ is strictly more specific than $\langle A_2, Q_2 \rangle$, nor $\langle A_2, Q_2 \rangle$ is strictly more specific than $\langle A_1, Q_1 \rangle$. In this case $\langle A_1, Q_1 \rangle$ is a blocking defeater of $\langle A, Q \rangle$.

Example 2.19 Consider Examples 2.13 and 2.15. Then $\langle B, \sim \text{fuel_ok} \rangle$ is a proper defeater for $\langle A, \text{engine_ok} \rangle$, since it is more specific.

This conceptualization allows us to apply the notion of counterargumentation (Definition 2.14) and defeat (Definition 2.18) in a natural way when assumption literals are involved, as shown in the following example.

Example 2.20 Consider a dlp $P = (\Pi, \Delta)$, where

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

Then $A = \{p \prec \text{not } q, r, \text{assume } \sim q\}$ 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 2.21 (Argumentation line) Let P be a dlp, and let $\langle A, Q \rangle$ be an argument in P . An argumentation line starting from $\langle A, Q \rangle$, denoted $\lambda^{\langle A, Q \rangle}$ (or simply λ) is a possibly infinite sequence of arguments

$$\lambda^{\langle A, Q \rangle} = [\langle A_0, Q_0 \rangle, \langle A_1, Q_1 \rangle, \langle A_2, Q_2 \rangle, \dots, \langle A_n, Q_n \rangle \dots]$$

satisfying the following conditions:

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

We distinguish two sets in any argumentation line λ : the set of supporting arguments $\lambda_S = \{ \langle A_0, Q_0 \rangle, \langle A_2, Q_2 \rangle, \langle A_4, Q_4 \rangle, \dots \}$ and the set of interfering arguments $\lambda_I = \{ \langle A_1, Q_1 \rangle, \langle A_3, Q_3 \rangle, \langle A_5, Q_5 \rangle, \dots \}$.

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 $\langle A_0, Q_0 \rangle$ 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 2.22 (Contradictory set of arguments) Given a dlp $P = (\Pi, \Delta)$, a set of arguments $S = \bigcup_{i=1}^n \{ \langle A_i, Q_i \rangle \}$ is contradictory wrt P iff $\Pi \cup \bigcup_{i=1}^n A_i$ is contradictory.

Definition 2.23 (Acceptable argumentation line) Let P be a dlp, and let $\lambda = [\langle A_0, Q_0 \rangle, \langle A_1, Q_1 \rangle, \dots, \langle A_n, Q_n \rangle, \dots]$ be an argumentation line in P . Let $\lambda' = [\langle A_0, Q_0 \rangle, \langle A_1, Q_1 \rangle, \dots, \langle A_k, Q_k \rangle, \dots]$ be an initial segment of λ . The sequence λ' is an acceptable argumentation line in P iff it is the longest initial segment in λ satisfying the following conditions:

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

The rationale for the conditions in Definition 2.23 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 2.24 Consider Example 2.9. The sequence

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

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

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

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

Proposition 2.25 Any acceptable argumentation line in a dlp P is finite.

Proof: Since P has no function symbols, and P is a finite set of program rules, the set of all possible arguments $Args(P)$ is necessarily finite. Hence the only way to get an infinite argumentation line $\lambda = [\langle A_0, Q_0 \rangle, \langle A_1, Q_1 \rangle, \langle A_2, Q_2 \rangle, \dots, \langle A_n, Q_n \rangle \dots]$ is by having the same argument twice in λ , i.e., $\langle A_i, Q_i \rangle = \langle A_j, Q_j \rangle$, and hence $A_i = A_j$. But this cannot be the case in an acceptable argumentation line because of condition 2 in Definition 2.23. Therefore any acceptable argumentation line λ is necessarily finite. ■

Let $\Lambda^{\langle A_0, Q_0 \rangle} = \{ \lambda_1, \lambda_2, \dots, \lambda_m \}$ be the set of all acceptable argumentation lines starting with $\langle A_0, Q_0 \rangle$ in a dlp P . A tree structure can be built out of the elements of $\Lambda^{\langle A_0, Q_0 \rangle}$, so that every path in the tree corresponds to some $\lambda_i \in \Lambda^{\langle A_0, Q_0 \rangle}$. This structure will be called *dialectical tree*. Formally:

Definition 2.26 (Dialectical tree) Let P be a dlp, and let A_0 be an argument for Q_0 in P . A dialectical tree for $\langle A_0, Q_0 \rangle$, denoted $T_{\langle A_0, Q_0 \rangle}$, is a tree structure defined as follows:

1. The root node of $T_{\langle A_0, Q_0 \rangle}$ is $\langle A_0, Q_0 \rangle$.
2. $\langle B', H' \rangle$ is an immediate child of $\langle B, H \rangle$ iff there exists an acceptable argumentation line $\lambda^{\langle A_0, Q_0 \rangle} = [\langle A_0, Q_0 \rangle, \langle A_1, Q_1 \rangle, \dots, \langle A_n, Q_n \rangle]$ such that there are two elements $\langle A_{i+1}, Q_{i+1} \rangle = \langle B', H' \rangle$ and $\langle A_i, Q_i \rangle = \langle B, H \rangle$, for some $i = 0, \dots, n-1$.

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 2.27.

Definition 2.27 (Marking of the dialectical tree) Let $\langle A, Q \rangle$ be an argument and $T_{\langle A, Q \rangle}$ its dialectical tree, then:

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

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

Definition 2.28 (Justification) Let A be an argument for a literal Q , and let $T_{\langle A, Q \rangle}$ be its associated acceptable dialectical tree. The argument A for Q will be a justification iff the root of $T_{\langle A, Q \rangle}$ is a U-node.

It can be shown [Gar97] that for any dlp P , strict arguments in P have no counterarguments, and therefore no defeaters. As a direct consequence of Definitions 2.26, 2.27 and 2.28, it follows that any strict argument 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 2.29 Consider Example 2.9, and assume our main query is *engine_ok*. An argument $\langle A, \text{engine_ok} \rangle$ can be built, which is defeated by the argument $\langle B, \sim \text{fuel_ok} \rangle$ (as shown in Examples 2.13, 2.15 and 2.19). Hence, the argument $\langle A, \text{engine_ok} \rangle$ will be provisionally rejected, since it is defeated. However, $\langle A, \text{engine_ok} \rangle$ can be reinstated, since there exists a third argument $C = \{ \sim \text{low_speed} \prec \text{sw2}, \text{sw1} \}$ for $\sim \text{low_speed}$ which on its turn defeats $\langle B, \sim \text{fuel_ok} \rangle$.

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

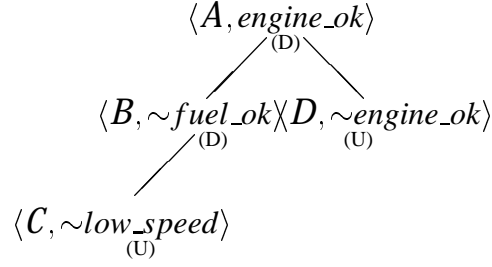


Figure 2: Dialectical tree (Example 2.9)

$pump_oil_ok \prec sw2$, $fuel_ok \prec pump_fuel_ok$, $oil_ok \prec pump_oil_ok$, $\sim engine_ok \prec \{fuel_ok, oil_ok, heat\}$. Hence $\langle D, engine_ok \rangle$ is once again provisionally defeated.

Since there are no more arguments to consider, $\langle A, engine_ok \rangle$ turns out to be ultimately defeated, so that we can conclude that the argument $\langle A, engine_ok \rangle$ 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 2.30 (Answers to a given query Q) Given a dlp P , a query Q can be classified as a positive, negative, undecided or unknown answer as follows:

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

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

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

Definition 2.31 (SEM_{DeLP})

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

$$\begin{aligned}
P_{accepted} &= \{Q \mid Q \in \text{Justified}(P)\} \\
P_{rejected} &= \{Q \mid Q \in \text{Unknown}(P) \cup \text{Negative}(P)\} \\
P_{undef} &= \{Q \mid Q \in \text{Undefined}(P)\}.
\end{aligned}$$

Example 2.32 Consider P as defined in Example 2.9, and consider the analysis performed in Example 2.29. Then $\text{engine_ok} \in \text{Negative}(P)$, $\sim\text{engine_ok} \in \text{Positive}(P)$, $\text{heat} \in \text{Positive}(P)$, and $\text{working_temperature_low} \in \text{Unknown}(P)$. Hence $\{\sim\text{engine_ok}, \text{heat}\} \subseteq P_{accepted}$, and $\text{engine_ok} \in P_{rejected}$.

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 $\text{SEM}(P_1) = \text{SEM}(P_2)$, for all P_1 and P_2 , such that $P_1 \mapsto P_2$. 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 P into a simplified version P' , from which the WFS can be easily read off.

Definition 3.1 (Transformation rules for WFS) Given a program $P \in \text{Prog}_L$, let $\text{HEAD}(P)$ be the set of all head-atoms of P , i.e., $\text{HEAD}(P) = \{H \mid H \leftarrow B^+, \text{not } B^-\}$. Let P_1 and P_2 be ground programs. The following transformation rules characterize WFS:

RED⁺ (Positive Reduction): Program P_2 results from program P_1 by **RED⁺** (written $P_1 \mapsto_P P_2$) iff there is a rule $H \leftarrow B$ in P_1 and a negative literal $\text{not } B \in B$ such that there is no rule about B in P_1 , i.e., $B \notin \text{HEAD}(P_1)$, and $P_2 = (P_1 \setminus \{H \leftarrow B\}) \cup \{H \leftarrow (B \setminus \{\text{not } B\})\}$.

RED⁻ (Negative Reduction): Program P_2 results from program P_1 by **RED⁻** (written $P_1 \mapsto_N P_2$) iff there is a rule $H \leftarrow B$ in P_1 and a negative literal $\text{not } B \in B$ such that B appears as a fact in P_1 , and $P_2 = P_1 \setminus \{H \leftarrow B\}$.

SUB (Deletion of non-minimal rules): Program P_2 results from program P_1 by **SUB** (written $P_1 \mapsto_M P_2$) iff there are rules $H \leftarrow B$ and $H \leftarrow B'$ in P_1 such that $B \subset B'$ and $P_2 = P_1 \setminus \{H \leftarrow B'\}$.

UNFOLD (Unfolding): Program P_2 results from program P_1 by **UNFOLD** (written $P_1 \mapsto_U P_2$) iff there is a rule $H \leftarrow B$ in P_1 and a positive literal $B \in B$ such that $P_2 = P_1 \setminus \{H \leftarrow B\} \cup \{H \leftarrow ((B \setminus \{B\}) \cup B') \mid B \leftarrow B' \in P_1\}$.

TAUT (Deletion of Tautologies): Program P_2 results from program P_1 by **TAUT** (written $P_1 \mapsto_T P_2$) iff there is $H \leftarrow B \in P_1$ such that $H \in B$ and $P_2 = P_1 \setminus \{H \leftarrow B\}$.

A program P' is a normal form of a program P wrt. a transformation “ \mapsto ” iff $P \mapsto_* P'$, where \mapsto_* denotes the reflexive-transitive closure of \mapsto , and P' is irreducible, i.e., there is no program P'' such that $P' \mapsto P''$.

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 P has a normal form P'), and *confluent* (i.e., given a program P , by applying the transformations in any fair order, we eventually arrive at a normal form $norm_{WFS}(P)$). This normal form $norm_{WFS}(P)$ is a *residual* program, consisting of rules without positive body atoms. For such a simplified program, its well-founded semantics can be easily read off as follows:

Definition 3.2 (SEM_{min}) We define $SEM_{min}(P) = \langle P^{true}, P^{false}, P^{undef} \rangle$ for any nlp P , where

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

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

Example 3.3 (Computing WFS) We consider the program P_1 and reduce it as follows:

$$\begin{array}{ccc} \begin{array}{l} p \\ q \leftarrow \text{not } p \\ q \leftarrow t, \text{not } p \\ s \leftarrow \text{not } q \\ q \leftarrow r \\ r \leftarrow q \end{array} & \xrightarrow{SUB} & \begin{array}{l} p \\ q \leftarrow \text{not } p \\ s \leftarrow \text{not } q \\ q \leftarrow r \\ r \leftarrow q \end{array} & \xrightarrow{RED^-} & \begin{array}{l} p \\ s \leftarrow \text{not } q \\ q \leftarrow r \\ r \leftarrow q \end{array} \end{array}$$

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

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

Now we can delete the resulting tautology by the application of **TAUT** and then use **Red⁺**

$$\begin{array}{ccc} \begin{array}{l} p \\ s \leftarrow \text{not } q \\ r \leftarrow q \end{array} & \xrightarrow{RED^+} & \begin{array}{l} p \\ s \\ r \leftarrow q \end{array} \end{array}$$

Finally applying **UNFOLD** to the last one, we get to $norm_{WFS}(P_1)$:

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

Thus, the wellfounded semantics of P_1 is:

$$WFS(P_1) = \{p, s, \text{not } q, \text{not } t, \text{not } r\}$$

Theorem 3.4 (Classifying WFS [BD99])

$$WFS(P) = SEM_{min}(norm_{WFS}(P)).$$

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 P (thus $HEAD(P)$ will stand for all heads of rules in 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 4.1 *Let P and P' be two dlp programs. If $Args(P) = Args(P')$, then $SEM_{DeLP}(P') = SEM_{DeLP}(P)$.*

Proof: This is a direct consequence of the Definition 2.31, since the semantics of *DeLP* is entirely determined by relationships among arguments. ■

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

Example 4.2 *Let $P_1 = \{ p \prec q, p \prec r, q \leftarrow, r \leftarrow \}$, and let $P_2 = \{ p \prec q, q \leftarrow, r \leftarrow \}$. Clearly, $SEM_{DeLP}(P_1) = SEM_{DeLP}(P_2)$, since $\{p, q, r\} = P_1^{accepted} = P_2^{accepted}$. However $Args(P_1) \neq Args(P_2)$ (since $\{p \prec r, p\}$ is an argument in P_1 but not in P_2).*

Definition 4.3 (Isomorphic dialectical trees) Given two arguments $\langle A_1, Q_1 \rangle$ and $\langle A_2, Q_2 \rangle$, their associated dialectical trees $T_{\langle A_1, Q_1 \rangle}$ and $T_{\langle A_2, Q_2 \rangle}$ will be isomorphic iff

1. $Q_1 = Q_2$, and both $\langle A_1, Q_1 \rangle$ and $\langle A_2, Q_2 \rangle$ have no defeaters, or
2. $T_{\langle A_1, Q_1 \rangle}$ has T_1, \dots, T_k as immediate subtrees, and $T_{\langle A_2, Q_2 \rangle}$ has T'_1, \dots, T'_k as immediate subtrees, and there exists a one-to-one correspondence $f : \{T_1, \dots, T_k\} \mapsto \{T'_1, \dots, T'_k\}$, such that
 - (a) T_i and $f(T_i)$ are isomorphic, $i = 1, \dots, k$, and
 - (b) The root of T_i is a proper (resp. blocking) defeater for $\langle A_1, Q_1 \rangle$ and the root of $f(T_i)$ is a proper (resp. blocking) defeater for $\langle A_2, Q_2 \rangle$, for $i = 1, \dots, k$.

Proposition 4.4 Let P_1 and P_2 be two $DeLP_{not}$ programs, such that $T_{\langle A_1, Q_1 \rangle}$ is the associated dialectical tree for an argument $\langle A_1, Q_1 \rangle$ in P_1 , and $T_{\langle A_2, Q_2 \rangle}$ is the associated dialectical tree for an argument $\langle A_2, Q_2 \rangle$ in P_2 . If $T_{\langle A_1, Q_1 \rangle}$ and $T_{\langle A_2, Q_2 \rangle}$ are isomorphic, then $Q_1 \in P_1^{accepted}$ (resp. $P_1^{rejected}$, P_1^{undef}) iff $Q_2 \in P_2^{accepted}$ (resp. $P_2^{rejected}$, P_2^{undef}).

Proof: This proposition is direct consequence of the definition of marking of a dialectical tree (Definition 2.27). ■

Corollary 4.5 Let P_1 and P_2 be two $DeLP_{not}$ programs, such that $HEAD(P_1) = HEAD(P_2)$. Suppose that for any literal Q in $HEAD(P_1)$, there exists a dialectical tree $T_{\langle A, Q \rangle}$ in P_1 iff there exists an isomorphic dialectical tree $T_{\langle B, Q \rangle}$ in P_2 . Then $SEM_{DeLP}(P_1) = SEM_{DeLP}(P_2)$.

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, P_1 and P_2 denote ground dlp programs. Some transformation rules have special requirements which appear in boldface.

RED_{neg}⁺: Program P_2 will result from program P_1 by **RED⁺** (written $P_1 \mapsto_{P_{neg}} P_2$) iff there is a rule $H \leftarrow B$ in P_1 and a negative literal $\sim B \in B$ such that there is no rule about B in P_1 , i.e., $B \notin HEAD(P_1)$, and $P_2 = (P_1 \setminus \{H \leftarrow B\}) \cup \{H \leftarrow (B \setminus \{\sim B\})\}$.

RED_{neg}⁻: Program P_2 will result from program P_1 by **RED⁻** (written $P_1 \mapsto_{M_{neg}} P_2$) iff there is a rule $H \leftarrow B$ in P_1 and a negative literal $\sim B \in B$ such that B appears as a fact in P_1 , and $P_2 = P_1 \setminus \{H \leftarrow B\}$.

SUB_{neg}: Program P_2 will result from program P_1 by **SUB** (written $P_1 \mapsto_{Sneg} P_2$) iff there are **strict** rules $H \leftarrow B$ and $H \leftarrow B'$ in P_1 such that $B \subset B'$ and $P_2 = P_1 \setminus \{H \leftarrow B'\}$. The rule $H \leftarrow B_2$ is called *non-minimal rule* wrt. $H \leftarrow B_1$.

UNFOLD_{neg}: Suppose program P_1 contains a **strict** rule $H \leftarrow B$ such that **there is no defeasible rule in P_1 with head H** .

Then program P_2 will result from program P_1 by **UNFOLD_{neg}** (written $P_1 \mapsto_{Uneg} P_2$) iff there is a positive literal $B \in B^2$ which **does not appear as head of a defeasible rule in P_1** , such that $P_2 = P_1 \setminus \{H \leftarrow B\} \cup \{H \leftarrow ((B \setminus \{B\}) \cup B') \mid B \leftarrow B' \in P_1\}$.

The clause $H \leftarrow B$ is said to be **UNFOLD_{neg}-related** with each $B \leftarrow B_i \in P_1$ (for $i = 1, \dots, n$).

TAUT_{neg}: Program P_2 will result from program P_1 by **TAUT_{neg}** (written $P_1 \mapsto_{Tneg} P_2$) iff there is $H \leftarrow B \in P_1$ such that $H \in B$ and $P_2 = P_1 \setminus \{H \leftarrow B\}$.

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

Example 4.6 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 P (since the argument $A_1 = \{ t \prec q_1 \}$ for p is defeated by the argument $A_2 = \{ \sim t \prec q_1, q_2 \}$ for $\sim t$). If we considered $P' = \mathbf{RED}_{neg}^+(P)$ we would get p as a fact, so p would be justified in P' .

Let us now consider **RED_{neg}⁻**. This transformation rule holds for both defeasible and strict rules in a $DeLP_{neg}$ program P , as shown in Proposition 4.7

Proposition 4.7 Let P be a $DeLP_{neg}$ program. Let P' be the resulting program of applying **RED_{neg}⁻**, i.e., $P \mapsto_{Mneg} P'$. Then $SEM_{DeLP}(P') = SEM_{DeLP}(P)$.

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

Let us now consider **SUB_{neg}**. This transformation holds for strict rules, as shown in Proposition 4.9. It does not hold in $DeLP_{neg}$ for defeasible rules (since having more literals in the body gives more specific information), as shown in Example 4.8

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

Example 4.8 Let $P = (\Pi, \Delta)$, 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 $A = \{(p \prec q_1, q_2)\}$ for p is strictly more specific than $B = \{(\sim p \prec q_2)\}$ for $\sim p$. However, if we consider $P' = P \setminus \{(p \prec q_1, q_2)\}$, then we get two arguments which block each other ($A = \{(p \prec q_1)\}$ for p and $B = \{(\sim p \prec q_2)\}$ for $\sim p$).

Proposition 4.9 Let P be a DeLP_{neg} program. Let P' be the program resulting from applying SUB_{neg} , i.e., $P \mapsto_{M_{neg}} P'$. Then $\text{SEM}_{\text{DeLP}}(P) = \text{SEM}_{\text{DeLP}}(P')$.

Proof: Clearly, $P = P \setminus \{r \mid r \text{ is a non-minimal rule}\}$. Let $r = P \leftarrow Q_1, \dots, Q_k$ be a non-minimal rule in P , and assume there is an argument A for some literal H in which r is part of the defeasible derivation for H . From the definition of defeasible derivation, for each literal Q_1, \dots, Q_k there is an argument $\langle B_1, Q_1 \rangle, \dots, \langle B_k, Q_k \rangle$, such that $\bigcup_{i=1}^k B_i \subseteq A$. Since r is a non-minimal rule, there exists $r' = P \leftarrow Q_1, \dots, Q_j \in \Pi$, $j < k$, such that for each literal Q_i ($i = 1, \dots, j$) there are arguments $\langle B_1, Q_1 \rangle, \dots, \langle B_j, Q_j \rangle$. But $\bigcup_{i=1}^j B_i \subseteq \bigcup_{i=1}^k B_k$. Hence by replacing r by r' we get either the same set A as an argument for H , or a proper subset $A' \subset A$ must be an argument for H . This means that A is not an argument according to Definition 2.10, because it does not satisfy condition 3. In any case, the rule r can be removed from P , without affecting the arguments that can be obtained from P . Therefore $\text{Args}(P) = \text{Args}(P')$ ($P' = P \setminus \{r\}$). Hence $\text{SEM}_{\text{DeLP}}(P) = \text{SEM}_{\text{DeLP}}(P')$. ■

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 4.10. 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 4.11.

Example 4.10 (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 P , there is an argument $A_1 = \{(\sim flies \prec bird, wounded), (bird \prec has_feathers, has_beak)\}$ for $\sim flies$ which is strictly more specific than $A_2 = \{(flies \prec bird), (bird \prec has_feathers, has_beak)\}$ for $flies$. In this case, the first argument is a justification. However, if UNFOLD_{neg} is applied on defeasible rules, we get $P' = (\Pi, \Delta')$, with $\Delta' = \{(flies \prec has_feathers, has_beak), (\sim flies \prec bird, wounded), (bird \prec has_feathers, has_beak)\}$. In P' we have two conflicting arguments, $A_1 = \{(\sim flies \prec bird, wounded), (bird \prec has_feathers, has_beak)\}$ for $\sim flies$ and

$A_2 = \{ (\text{flies} \prec \text{has_feathers}, \text{has_beak}) \}$ for *flies*. In this case, neither of them is strictly more specific than the other.

Example 4.11 Let $P = (\Pi, \Delta)$ be a dlp, where $\Pi = \{ (p \leftarrow q, s), (q \leftarrow f_1), (q \leftarrow f_2), (s \leftarrow) \}$, and $\Delta = \{ q \prec s \}$. If we could apply UNFOLD_{neg} on rule $(p \leftarrow q, s)$ wrt. the literal q , we would get the program $P' = P \setminus \{(p \leftarrow q, s)\} \cup \{(p \leftarrow f_1, s), (p \leftarrow f_2, s)\}$. But $A_1 = \{ q \prec s \}$ is an argument for p in P , but it does not exist in P' .

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

Definition 4.12 (Transformation $\text{UNFOLD}_{neg}^{r_i}$) Suppose program P_1 contains a strict rule $H \leftarrow B$ such that **there is no defeasible rule in P_1 with head H** .

Then program P_2 will result from program P_1 by $\text{UNFOLD}_{neg}^{r_i}$ (written $P_1 \mapsto_{U_{neg}}^{r_i} P_2$) iff there is a positive literal $B \in \mathcal{B}$ which **does not appear as head of a defeasible rule in P_1** , such that $P_2 = P_1 \setminus \{H \leftarrow B\} \cup \{H \leftarrow ((B \setminus \{B\}) \cup B') \mid r_i = B \leftarrow B' \in P_1\}$.

Proposition 4.13 Let P_1 be a DeLP_{neg} program which contains a strict rule $r = H \leftarrow B$, such that r_1, r_2, \dots, r_k are all those rules in P_1 that are UNFOLD_{neg} -related to r . Consider the sequence of programs $P = P_1 \mapsto_{U_{neg}}^{r_1} P_2 \mapsto_{U_{neg}}^{r_2} \dots, P_k \mapsto_{U_{neg}}^{r_k} P'$. Then $P \mapsto_{U_{neg}} P'$ wrt rule r .

Proof: Direct consequence of Definition 4.12 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 $\mapsto_{U_{neg}}^{r_i}$ preserves semantics when applied to a given DeLP_{neg} program.

Proposition 4.14 Let $\langle A, H \rangle$ be an argument in DeLP_{neg} , such that the last rule used in the derivation is the strict rule $H \leftarrow P_1, \dots, P_k$. Then all immediate subarguments $\langle A_1, P_1 \rangle, \dots, \langle A_k, P_k \rangle$ are such that $A_i = A, \forall i = 1, \dots, k$.

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

Proposition 4.15 Let P_1 be a DeLP_{neg} program, and let P_2 be the program resulting from applying $\mapsto_{U_{neg}}^{r_i}$ wrt some rule r_i .

Let $\langle A, H \rangle$ be an argument in P_1 affected by the application of $\mapsto_{U_{neg}}^{r_i}$. Then $\langle A, H \rangle$ is also an argument in P_2 , and $\text{Args}(P_1) = \text{Args}(P_2)$.

Proof: Let $P_1 = (\Pi, \Delta)$ be a $DeLP_{neg}$ program, and let $\langle A, Q \rangle$ be an argument in P_1 , such that (i) a strict rule $r = H \leftarrow B$ is used in the defeasible derivation of Q from $\Pi \cup A$, and (ii) r is $UNFOLD_{neg}$ -related to other rule r_i . If this is not the case, then clearly $Args(P_1) = Args(P_2)$, and the proposition holds trivially.

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

$$r = H \leftarrow B, L_1, \dots, L_k \quad (1)$$

From Proposition 4.14, we get that $\langle S, B \rangle, \langle S, L_1 \rangle, \dots, \langle S, L_k \rangle$ are immediate subarguments of $\langle S, H \rangle$.

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

$$r_i = B \leftarrow P_1, \dots, P_m. \quad (2)$$

From Proposition 4.14, we get that $\langle S, P_1 \rangle, \langle S, P_2 \rangle, \dots, \langle S, P_m \rangle$ are immediate subarguments of $\langle S, B \rangle$. Thus, argument $\langle S, H \rangle$ in P_1 is such that $\langle S, P_1 \rangle, \dots, \langle S, P_m \rangle$ and $\langle S, L_1 \rangle, \dots, \langle S, L_k \rangle$ are also arguments in P_1 .

Assume we apply $\mapsto_{Uneg}^{r_i}$ to P_1 , resulting in a new $DeLP_{neg}$ program P_2 . From Definition 4.12, we have:

$$P_2 = P_1 \setminus \{H \leftarrow B\} \cup \{H \leftarrow ((B \setminus \{B\}) \cup B') \mid r_i = B \leftarrow B' \in P_1\}$$

In this case we get $P_2 = P_1 \setminus \{r\} \cup \{r'\}$, where r' is the rule

$$r' = H \leftarrow L_1, \dots, L_k, P_1, \dots, P_m \quad (3)$$

Clearly, $\langle S, L_i \rangle, i = 1, \dots, k$ and $\langle S, P_i \rangle, i = 1, \dots, m$ are also arguments in P_2 , and in particular $\langle S, H \rangle$ is also an argument in P_2 . Note that no new argument other than $\langle S, H \rangle$ is generated in P_2 , since the subarguments of $\langle S, H \rangle$ in P_1 and $\langle S, H \rangle$ in P_2 are the same. Thus $Args(P_1) = Args(P_2)$. ■

Corollary 4.16 *Let P be a $DeLP_{neg}$ program, and let P' be the program resulting from applying $UNFOLD_{neg}$ wrt some rule r in P . Then $SEM_{DeLP}(P) = SEM_{DeLP}(P')$.*

Proof: Follows directly from Proposition 4.13 by repeated application of $\mapsto_{Uneg}^{r_i}$, for each r_i which is $UNFOLD_{neg}$ -related with r . ■

Let us now consider tautology elimination.

Proposition 4.17 *Let P be a $DeLP_{neg}$ program, and P' the program resulting from applying $TAUT_{neg}$ to P , i.e., $P \mapsto_{Tneg} P'$ Then $SEM_{DeLP}(P) = SEM_{DeLP}(P')$.*

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

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 SUB_{neg} and $UNFOLD_{neg}$. 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 SUB_{not} and $UNFOLD_{not}$, as presented in Section 4.2.

4.2 Transformation Properties in $DeLP_{not}$

$DeLP_{not}$ 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 $\langle A, Q_1 \rangle$ accounts for a *counterargument* for an argument $\langle B, Q_2 \rangle$ if $not Q_1$ is used as an *assumption* in the defeasible derivation of Q_2 from $\Pi \cup B$.

Assumption literals are the only possible points for attack in $DeLP_{not}$. In fact, we now restrict our framework in that we allow in Definition 2.10 only assume $\sim A$ where A is an atom. That is, we do not allow assume A literals. Thus the set $H_{\langle A, Q \rangle}$ denotes in this section the set of assumption literals in $\langle A, Q \rangle$ 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 2.6 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 4.18 *Let P be a $DeLP_{not}$ program, and let $\langle A, Q \rangle$ be an argument in P such that Q follows from A using assume $\sim R$ as an assumption. If $\langle \emptyset, R \rangle$, then $\langle A, Q \rangle$ is not a justification.*

Proof: Clearly $\langle \emptyset, R \rangle$ is a counterargument for $\langle A, Q \rangle$, in particular (according to specificity) a defeater. Since $\langle \emptyset, R \rangle$ has no defeaters (as discussed on page 11), the dialectical tree with root $\langle A, Q \rangle$ will have a children node $\langle \emptyset, R \rangle$, which will turn out to be marked as U (according to Definitions 2.27). Hence $\langle A, Q \rangle$ will be marked as D , so that $\langle A, Q \rangle$ is not a justification. ■

The precise semantics for $DeLP_{not}$ depends on the analogue of Definitions 2.14 and 2.18 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}_{not}^r , \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 $P \mapsto_{R^+not} P'$ (resp. \mapsto_{R^-not} , \mapsto_{Snot} , \mapsto_{Unot} , \mapsto_{Unot}^r , \mapsto_{Tnot}) to denote the $DeLP_{not}$ program P' resulting from P by application of the transformation \mathbf{RED}_{not}^+ (resp. \mathbf{RED}_{not}^- , \mathbf{SUB}_{not} , \mathbf{UNFOLD}_{not} , \mathbf{UNFOLD}_{not}^r , \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 on defeasible rules (as shown in Examples 4.10 and 4.8).

Proposition 4.19 *Let P be a $DeLP_{not}$ program. Let P' be the $DeLP_{not}$ program resulting from $P \mapsto_{R^+not} P'$. Then $SEM_{DeLP}(P) = SEM_{DeLP}(P')$.*

Proof: Let P be a $DeLP_{not}$ program, such that $r = P \multimap Q_1, \dots, not Q, \dots, Q_k$ is a defeasible rule in P , and there is no rule about Q in P . Let P' be the $DeLP_{not}$ program resulting from applying \mapsto_{R^+not} to P on rule r .

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

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

Hence each dialectical tree T in P involving $\langle B, S \rangle$ as a node is isomorphic to T' in P' involving $\langle B', S \rangle$ in P' . From Proposition 4.4 it follows that $SEM_{DeLP}(P) = SEM_{DeLP}(P')$. ■

Proposition 4.20 *Let P be a DeLP_{not} program. Let P' be the DeLP_{not} program resulting from $P \mapsto_{R^{-not}} P'$. Then $SEM_{DeLP}(P) = SEM_{DeLP}(P')$.*

Proof: Let $P = (\Pi, \Delta)$ be a DeLP_{not} program. Let $r = P \leftarrow Q_1, \dots, \text{not } Q, \dots, Q_n$ be a strict rule, and assume $Q \leftarrow \in P$. Assume r is used in a defeasible derivation for building an argument $\langle A, H \rangle$. Clearly $\Pi \cup A \vdash Q$ and $\Pi \cup A \vdash \text{assume } \sim Q$. But this violates condition 2 in Definition 2.10 (contradiction). Therefore each argument $\langle A, H \rangle$ in P is also an argument in $P \setminus \{r\}$. Hence $Args(P) = Args(P')$, and therefore $SEM_{DeLP}(P) = SEM_{DeLP}(P')$. ■

Proposition 4.21 *Let P be a DeLP_{not} program. Let P' be the DeLP_{not} program resulting from $P \mapsto_{Snot} P'$. Then $SEM_{DeLP}(P) = SEM_{DeLP}(P')$.*

Proof: Let P be a DeLP_{not} program, and let $r = P \leftarrow B_l$ be a non-minimal strict rule in P (i.e., there exists a rule $r' = P \leftarrow B_2$ such that $B_2 \subseteq B_l$). We consider $B_l = B_1^+ \cup \text{not } B_1^-$, distinguishing the set B_1^+ of positive literals from the set $\text{not } B_1^-$ (literals preceded by not). If $B_2 \subseteq B_l$, then two situations are to be considered: either $B_2^+ \subseteq B_1^+$, or $B_2^- \subseteq B_1^-$.

1. Suppose $B_2^+ \subseteq B_1^+$. Then $Args(P) = Args(P \setminus \{r\})$, following the same line of reasoning as in Proposition 4.9.
2. Suppose that $B_2^- \subseteq B_1^-$, $B_2^+ = B_1^+$. Suppose there exists an argument $\langle A, H \rangle$ such that the strict rule $r = P \leftarrow B_l$ is used in the defeasible derivation of H . Clearly, there is an assumption literal $\text{assume } \sim Q$ in A for each $\text{not } Q$ in B_1^- . Let H_1 be the set of assumption literals in A . It follows that $A \setminus H_2$ also provides a defeasible derivation for H using r' instead, where H_2 is the set of assumption literals in r' , such that $H_2 \subseteq H_1$. But then the defeasible derivation of H using r violates condition 3 in Definition 2.10. Therefore no argument using r can be built in P , so that $Args(P) = Args(P \setminus \{r\})$.

From this analysis it follows that $P \mapsto_{Snot} P'$ is such that $SEM_{DeLP}(P) = SEM_{DeLP}(P')$. ■

Proposition 4.22 *Let P be a DeLP_{not} program. Let P' be the DeLP_{not} program resulting from $P \mapsto_{Tnot} P'$. Then $SEM_{DeLP}(P) = SEM_{DeLP}(P')$.*

Proof: We will consider only the case in which literals preceded by *not* are present in a rule of the form $r = P \leftarrow P, Q_1, \dots, Q_k$. Otherwise the proof follows the same line of reasoning as in Proposition 4.17.

1. Suppose there exists an argument $\langle A, H \rangle$ in P such that $\Pi \cup A \vdash H$ using a strict rule $r = P \leftarrow P, Q_1, \dots, \text{not } Q, \dots, Q_k$. Then the occurrence of P in the antecedent of r can also be proven from $\Pi \setminus \{r\} \cup A'$, where $A' = A \setminus \{\text{assume } \sim Q\}$. But then $\langle A, H \rangle$ is not an argument, since it violates condition 3 in Definition 2.10.

Therefore, no rule $r = P \leftarrow P, Q_1, \dots, \text{not } Q, \dots, Q_k$ can be used in an argument in P . Hence $\text{Args}(P) = \text{Args}(P')$, with $P' = P \setminus \{r\}$ so that $\text{SEM}_{\text{DeLP}}(P) = \text{SEM}_{\text{DeLP}}(P')$.

2. Suppose there exists an argument $\langle A, H \rangle$ in P such that $\Pi \cup A \vdash H$ using a defeasible rule $r = P \prec P, Q_1, \dots, \text{not } Q, \dots, Q_k$. The same line of reasoning as above applies, with $A' = A \setminus \{r, \text{assume } \sim Q\}$. Therefore $\text{SEM}_{\text{DeLP}}(P) = \text{SEM}_{\text{DeLP}}(P')$.

■

We present next a property of immediate subarguments in DeLP_{not} , similar to the one shown in Proposition 4.14. Then we will show that the transformation $\mapsto_{U_{\text{not}}}^{r_i}$ preserves semantics when applied to a given DeLP_{not} program.

Proposition 4.23 *Let $\langle A, H \rangle$ be an argument in DeLP_{not} , such that the last rule used in the derivation is the strict rule $H \leftarrow P_1, \dots, P_k, \text{not } L_1, \dots, \text{not } L_j$, distinguishing literals from assumption literals. Then all immediate subarguments $\langle A_1, P_1 \rangle, \dots, \langle A_k, P_k \rangle$ are such that $A_i = A \setminus \bigcup_{i=1}^j \{\text{assume } \sim L_i\}$, $\forall i = 1, \dots, k$.*

Proof: Follows from the same line of reasoning in Proposition 4.14 when considering strict rules with assumption literals. ■

Proposition 4.24 *Let P be a DeLP_{not} program. Let P' be the DeLP_{not} program resulting from $P \mapsto_{U_{\text{not}}}^{r_i} P'$ wrt a strict rule r in P . Then $\text{SEM}_{\text{DeLP}}(P) = \text{SEM}_{\text{DeLP}}(P')$.*

Proof: Let $P_1 = (\Pi, \Delta)$ be a DeLP_{not} program, and let $\langle A, Q \rangle$ be an argument in P_1 , such that (i) a strict rule $r = H \leftarrow B$ is used in the defeasible derivation of Q from $\Pi \cup A$, and (ii) r is $\text{UNFOLD}_{\text{not}}$ -related to other rule r_i . If this is not the case, then clearly $\text{Args}(P_1) = \text{Args}(P_2)$, and the proposition holds trivially. We can also assume that $\emptyset \subset B^- \subseteq B$, i.e., there is at least one literal preceded by *not* in B ; otherwise the proposition follows directly from Proposition 4.15.

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

The strict rule r can be written as

$$r = H \leftarrow B, L_1, \dots, L_k, \text{not } M_1, \dots, \text{not } M_j \quad (4)$$

distinguishing positive literals from literals preceded by *not*. Let $S_1 = S \setminus \bigcup_{i=1}^j \{\text{assume } \sim M_i\}$. From Proposition 4.23, we get that $\langle S_1, B \rangle, \langle S_1, L_1 \rangle, \dots, \langle S_1, L_k \rangle$ are immediate subarguments of $\langle S, H \rangle$. Hence we get that

$$H_{\langle S, H \rangle} = H_{\langle S_1, B \rangle} \cup \bigcup_{i=1}^k H_{\langle S_1, L_i \rangle} \cup \bigcup_{i=1}^j \{\text{assume } \sim M_i\} \quad (5)$$

Consider $r_i = B \leftarrow B$, which is the last rule used in the defeasible derivation of $\langle S_1, B \rangle$, such that r is **UNFOLD**_{not}-related to r_i . Since r_i is an arbitrary strict rule, it will have the form

$$r_i = B \leftarrow P_1, \dots, P_m, \text{not} R_1, \dots, \text{not} R_p. \quad (6)$$

Let $S_2 = S_1 \setminus \bigcup_{i=1}^p \{\text{assume } \sim R_i\}$ It follows that

$$H_{\langle S_1, B \rangle} = \bigcup_{i=1}^m H_{\langle S_2, P_i \rangle} \cup \bigcup_{j=1}^p \{\text{assume } \sim R_j\} \quad (7)$$

Replacing (7) in (5), we get

$$H_{\langle S, H \rangle} = \bigcup_{i=1}^m H_{\langle S_2, P_i \rangle} \cup \bigcup_{j=1}^p \{\text{assume } \sim R_j\} \cup \bigcup_{i=1}^k H_{\langle S_1, L_i \rangle} \cup \bigcup_{i=1}^j \{\text{assume } \sim M_i\} \quad (8)$$

Thus, argument $\langle S, H \rangle$ in P_1 is such that $S = R_S \cup H_{\langle S, H \rangle}$, where $H_{\langle S, H \rangle}$ is defined as in (8). Assume we apply $\mapsto_{\text{UNFOLD}_{not}}$ to P_1 , where the rule r is **UNFOLD**_{not}-related to r_i , resulting in a new *DeLP*_{not} program P_2 . From the definition of **UNFOLD**_{not} ^{r_i} , we have:

$$P_2 = P_1 \setminus \{H \leftarrow B\} \cup \{H \leftarrow ((B \setminus \{B\}) \cup B') \mid r_i = B \leftarrow B' \in P_1\}$$

Consider the original rule r in (4), and the **UNFOLD**_{not}-related rule r_i in (6). Let P_2 be the *DeLP*_{not} program resulting from applying the UNFOLD transformation to r with respect to r_i . In this case we get

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

or equivalently

$$H \leftarrow L_1, \dots, L_k, P_1, \dots, P_m, \text{not} M_1, \dots, \text{not} M_j, \text{not} R_1, \dots, \text{not} R_p \quad (9)$$

Let $S' = S \setminus \{\bigcup_{i=1}^j \{\text{assume } \sim M_i\} \cup \bigcup_{i=1}^p \{\text{assume } \sim R_i\}\}$. From Proposition 4.23, it follows that $\langle S', L_i \rangle$, $i = 1, \dots, k$ and $\langle S', P_i \rangle$, $i = 1, \dots, m$ are arguments in P_2 . In particular, we have

$$H_{\langle S', H \rangle} = \bigcup_{i=1}^k H_{\langle S', L_i \rangle} \cup \bigcup_{i=1}^m H_{\langle S', P_i \rangle} \cup \bigcup_{j=1}^p \{\text{assume } \sim R_j\} \cup \bigcup_{i=1}^j \{\text{assume } \sim M_i\} \quad (10)$$

Hence $R_S \cup H_{\langle S', H \rangle}$ is an argument for H in P_2 , since every defeasible rule in P_1 is also a defeasible rule in P_2 . But from (8) and (10) it follows that $H_{\langle S', H \rangle} = H_{\langle S, H \rangle}$, and the set $S' = S$. Hence, $\langle S, H \rangle$ is an argument in both P_1 and P_2 .

	<i>NLP</i> under wfs	<i>DeLP_{neg}</i>	<i>DeLP_{not}</i>
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

Figure 3: Behavior of *NLP*, *DeLP_{neg}* and *DeLP_{not}* under different transformations

^aSome additional conditions are required for the transformation to hold.

Therefore, we can conclude that for any argument $\langle S, H \rangle$ in 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 $\langle S, H \rangle$ is also an argument in P_2 . Note that no new argument other than $\langle S, H \rangle$ is generated in P_2 , since the subarguments of $\langle S, H \rangle$ in P_1 and $\langle S, H \rangle$ in P_2 are the same. Hence $Args(P_1) = Args(P_2)$, and therefore $SEM_{DeLP}(P_1) = SEM_{DeLP}(P_2)$. ■

Corollary 4.25 *Let P be a *DeLP_{not}* program. Let P' be the *DeLP_{not}* program resulting from $P \mapsto_{U_{not}} P'$ wrt a strict rule r in P . Then $SEM_{DeLP}(P) = SEM_{DeLP}(P')$.*

Proof: Follows directly from Proposition 4.13 by repeated application of $\mapsto_{U_{not}}^{r_i}$, for each r_i which is **UNFOLD_{not}**-related with r . ■

4.3 Relating *NLP* and *DeLP_{not}* under WFS

A natural question is how well-founded semantics WFS relates to *DeLP_{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 4.26 (DeLP_{not} extends WFS) *Let P be a program in *NLP*. We can look at P also as a theory in *DeLP_{not}*. Then all atoms A and default atoms $notA$ that are true in $WFS(P)$ are also contained in $SEM_{DeLP_{not}}(P)$.*

Proof: As all the transformation properties hold, we can transform P into a normalform where all rules only have negative body literals (or are empty):

- The atoms true in $WFS(P)$ are, by Theorem 3.4, exactly those A where there is a rule of the form “ $A \leftarrow$ ”. But those atoms are certainly justified in $SEM_{DeLP_{not}}(P)$.
- All default literals $notA$ that are true in $WFS(P)$ are, by Theorem 3.4, exactly those A where there is no rule with head A . But then assume $\sim A \leftarrow$ can be assumed as it can not lead to any contradiction. ■

Example 4.27 Consider the normal logic programs

$$\begin{aligned}
P_1 &= \{ (a \leftarrow b), (b \leftarrow a), (c \leftarrow \text{nota}, \text{not}b) \} \\
P_2 &= \{ (a \leftarrow \text{not}b), (b \leftarrow a) \} \\
P_3 &= \{ (a \leftarrow \text{not}b), (b \leftarrow \text{nota}), (c \leftarrow a), (c \leftarrow b) \} \\
P_4 &= \{ (a \leftarrow b, \text{not}d), (b \leftarrow a, \text{not}d), (d \leftarrow \text{not}d), (c \leftarrow \text{nota}, \text{not}b) \} \\
P_5 &= \{ (a \leftarrow \text{not}b), (b \leftarrow \text{nota}), (a \leftarrow \text{nota}) \}
\end{aligned}$$

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

- WFS in P_1 is $\{\text{nota}, \text{not}b, c\}$. The only argument that can be constructed from P_1 as a DeLP_{not} program is the one which justifies c .
Without the last rule $(c \leftarrow \text{nota}, \text{not}b)$ no arguments for positive atoms can be constructed.
- WFS in P_2 is empty. Under DeLP_{not} , no argument can be built, since the only possible set $\{\text{assume } \sim b\}$ leads to contradiction.
- WFS in P_3 is empty. In DeLP_{not} , two sets of assumptions are possible for building arguments: $A_1 = \{\text{assume } \sim a\}$ and $A_2 = \{\text{assume } \sim b\}$. We can build the arguments $\langle A_1, b \rangle$, $\langle A_2, a \rangle$, $\langle A_1, c \rangle$, $\langle A_2, c \rangle$. Any one of these arguments has a blocking defeater. From Definition 2.28 it follows that no argument is justified.
- WFS in P_4 is $\{\text{nota}, \text{not}b, c\}$. The only argument that can be constructed from P_4 as a DeLP_{not} program is the one which justifies c .
However, without the last rule $c \leftarrow \text{nota}, \text{not}b$ no argument can be built in P_4 under DeLP_{not} (there is no defeasible sequence for a nor for b).
- WFS in P_5 is empty. But in $\text{SEM}_{\text{DeLPnot}}$ the argument $\{\text{assume } \sim b\}$ is a justification for a . This is because $\langle \{\text{assume } \sim b\}, a \rangle$ cannot be defeated (the only way to do this would be to find an argument involving the assumption $\text{not}a$, but this would lead to a contradiction).

The last program P_5 shows that $\text{SEM}_{\text{DeLPnot}}$ 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 **RED⁻**). It is worth noticing that **RED⁻** holds in *NLP* because of a “consistency constraint” (it cannot be the case that both *not P* and *P* hold). The same is achieved in *DeLP* by demanding non-contradiction when constructing arguments.
- Other transformation properties only hold for strict rules (e.g. **SUB**), sometimes with extra requirements (e.g. **UNFOLD**). This shows that defeasible rules express a link between literals that cannot be easily “simplified” in terms of a transformation rule, and a more complex analysis (e.g. computing defeat) is required.
- Some properties (e.g. **RED⁺**) do not hold at all wrt. strict negation, but do hold wrt. default negation. In the first case, the reason is that negated literals are treated as new predicate names (and succeed as subgoals iff they can be proven from the program). In the second case, default negation behaves much like its counterpart in *NLP*. As in *NLP*, the absence of rules with head *H* is enough for concluding that *H* cannot be proven, and therefore not justified.

5 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 *not L* should hold too (similarly, if *L* then $\sim L$). 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 **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. (Extended abstract appeared in: Characterizations of the Stable Semantics by Partial Evaluation *LPNMR, Proceedings of the Third International Conference, Kentucky*, pages 85–98, 1995. LNCS 928, Springer.).
- [BD98] S. Brass and J. Dix. Characterizations of the Disjunctive Well-founded Semantics: Confluent Calculi and Iterated GCWA. *Journal of Automated Reasoning*, 20(1):143–165, 1998.
- [BD99] S. Brass and J. Dix. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, 38(3):167–213, 1999.
- [BD01] Gerhard Brewka and Jürgen Dix. Knowledge representation with extended logic programs. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic, 2nd Edition, Volume 6, Methodologies*, chapter 6.

- Reidel Publ., 2001. Shortened version also appeared in Dix, Pereira, Przymusinski (Eds.), *Logic Programming and Knowledge Representation*, Springer LNAI 1471, pages 1–55, 1998.
- [BDFZ01] Stefan Brass, Juergen Dix, Burkhard Freitag, and Ulrich Zukowski. Transformation-based Bottom-up Computation of the Well-Founded Model. *Theory and Practice of Logic Programming*, 2001. To appear.
- [BDKT97] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997.
- [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*. 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*, to appear, 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. Fillotrani. 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 Nonmonotonic 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.
- [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.
- [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.
- [Vre93] Gerard A.W. Vreeswijk. *Studies in Defeasible Argumentation*. PhD thesis, Vrije University, Amsterdam (Holland), 1993.

Available Research Reports (since 1998):

2001

- 3/2001** *Carlos I. Chesñevar, Jürgen Dix, Frieder Stolzenburg, Guillermo R. Simari.* Relating Defeasible and Normal Logic Programming through Transformation Properties.
- 2/2001** *Carola Lange, Harry M. Sneed, Andreas Winter.* Applying GUPRO to GEOS – A Case Study.
- 1/2001** *Pascal von Hutten, Stephan Philippi.* Modelling a concurrent ray-tracing algorithm using object-oriented Petri-Nets.

2000

- 8/2000** *Jürgen Ebert, Bernt Kullbach, Franz Lehner (Hrsg.).* 2. Workshop Software Reengineering (Bad Honnef, 11./12. Mai 2000).
- 7/2000** *Stephan Philippi.* AWPN 2000 - 7. Workshop Algorithmen und Werkzeuge für Petrinetze, Koblenz, 02.-03. Oktober 2000 .
- 6/2000** *Jan Murray, Oliver Obst, Frieder Stolzenburg.* Towards a Logical Approach for Soccer Agents Engineering.
- 5/2000** *Peter Baumgartner, Hantao Zhang (Eds.).* FTP 2000 – Third International Workshop on First-Order Theorem Proving, St Andrews, Scotland, July 2000.
- 4/2000** *Frieder Stolzenburg, Alejandro J. García, Carlos I. Chesñevar, Guillermo R. Simari.* Introducing Generalized Specificity in Logic Programming.
- 3/2000** *Ingar Uhe, Manfred Rosendahl.* Specification of Symbols and Implementation of Their Constraints in JKogge.
- 2/2000** *Peter Baumgartner, Fabio Massacci.* The Taming of the (X)OR.
- 1/2000** *Richard C. Holt, Andreas Winter, Andy Schürr.* GXL: Towards a Standard Exchange Format.

1999

- 10/99** *Jürgen Ebert, Luuk Groenewegen, Roger Süttenbach.* A Formalization of SOCCA.
- 9/99** *Hassan Diab, Ulrich Furbach, Hassan Tabbara.* On the Use of Fuzzy Techniques in Cache Memory Management.
- 8/99** *Jens Woch, Friedbert Widmann.* Implementation of a Schema-TAG-Parser.

- 7/99** *Jürgen Ebert, and Bernt Kullbach, Franz Lehner (Hrsg.).* Workshop Software-Reengineering (Bad Honnef, 27./28. Mai 1999).
- 6/99** *Peter Baumgartner, Michael Kühn.* Abductive Coreference by Model Construction.
- 5/99** *Jürgen Ebert, Bernt Kullbach, Andreas Winter.* GraX – An Interchange Format for Reengineering Tools.
- 4/99** *Frieder Stolzenburg, Oliver Obst, Jan Murray, Björn Bremer.* Spatial Agents Implemented in a Logical Expressible Language.
- 3/99** *Kurt Lautenbach, Carlo Simon.* Erweiterte Zeitstempelnetze zur Modellierung hybrider Systeme.
- 2/99** *Frieder Stolzenburg.* Loop-Detection in Hyper-Tableaux by Powerful Model Generation.
- 1/99** *Peter Baumgartner, J.D. Horton, Bruce Spencer.* Merge Path Improvements for Minimal Model Hyper Tableaux.

1998

- 24/98** *Jürgen Ebert, Roger Süttenbach, Ingar Uhe.* Meta-CASE Worldwide.
- 23/98** *Peter Baumgartner, Norbert Eisinger, Ulrich Furbach.* A Confluent Connection Calculus.
- 22/98** *Bernt Kullbach, Andreas Winter.* Querying as an Enabling Technology in Software Reengineering.
- 21/98** *Jürgen Dix, V.S. Subrahmanian, George Pick.* Meta-Agent Programs.
- 20/98** *Jürgen Dix, Ulrich Furbach, Ilkka Niemelä .* Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations.
- 19/98** *Jürgen Dix, Steffen Hölldobler.* Inference Mechanisms in Knowledge-Based Systems: Theory and Applications (Proceedings of WS at KI '98).
- 18/98** *Jose Arrazola, Jürgen Dix, Mauricio Osorio, Claudia Zepeda.* Well-behaved semantics for Logic Programming.
- 17/98** *Stefan Brass, Jürgen Dix, Teodor C. Przymusiński.* Super Logic Programs.
- 16/98** *Jürgen Dix.* The Logic Programming Paradigm.

- 15/98** *Stefan Brass, Jürgen Dix, Burkhard Freitag, Ulrich Zukowski.* Transformation-Based Bottom-Up Computation of the Well-Founded Model.
- 14/98** *Manfred Kamp.* GReQL – Eine Anfragesprache für das GUPRO-Repository – Sprachbeschreibung (Version 1.2).
- 12/98** *Peter Dahm, Jürgen Ebert, Angelika Franzke, Manfred Kamp, Andreas Winter.* TGraphen und EER-Schemata – formale Grundlagen.
- 11/98** *Peter Dahm, Friedbert Widmann.* Das Graphenlabor.
- 10/98** *Jörg Jooss, Thomas Marx.* Workflow Modeling according to WfMC.
- 9/98** *Dieter Zöbel.* Schedulability criteria for age constraint processes in hard real-time systems.
- 8/98** *Wenjin Lu, Ulrich Furbach.* Disjunctive logic program = Horn Program + Control program.
- 7/98** *Andreas Schmid.* Solution for the counting to infinity problem of distance vector routing.
- 6/98** *Ulrich Furbach, Michael Kühn, Frieder Stolzenburg.* Model-Guided Proof Debugging.
- 5/98** *Peter Baumgartner, Dorothea Schäfer.* Model Elimination with Simplification and its Application to Software Verification.
- 4/98** *Bernt Kullbach, Andreas Winter, Peter Dahm, Jürgen Ebert.* Program Comprehension in Multi-Language Systems.
- 3/98** *Jürgen Dix, Jorge Lobo.* Logic Programming and Nonmonotonic Reasoning.
- 2/98** *Hans-Michael Hanisch, Kurt Lautenbach, Carlo Simon, Jan Thieme.* Zeitstempelnetze in technischen Anwendungen.
- 1/98** *Manfred Kamp.* Managing a Multi-File, Multi-Language Software Repository for Program Comprehension Tools — A Generic Approach.