International Journal of Foundations of Computer Science

ON THE PROPOSITIONAL SLDNF-RESOLUTION

JAN A. PLAZA
University of Miami
Department of Mathematics and Computer Science
P.O. Box 249085, Coral Gables, Florida 33124, U.S.A.
janplaza@math.cs.miami.edu
http://www.cs.miami.edu/~janplaza

ABSTRACT

We consider propositional logic programs with negations. We define notions of constructive transformation and constructive completion of a program. We use these notions to characterize SLDNF-resolution in classical, intuitionistic and intermediate logics, and also to derive a characterization in modal logics of knowledge. We show that the three-valued and four-valued fix-point or declarative semantics for program P are equivalent to the two-valued semantics for the constructive version of P.
We argue that it would be beneficial to replace Negation as Failure by constructive transformation, and it would be beneficial to use the two-valued semantics for the constructive version of the program instead of multivalued semantics for the original program.

Keywords: Logic programming, declarative programming, negation as failure, non-classical logics, three-valued semantics.

1. Introduction

This paper is concerned with declarative interpretations of logic programs using several logical systems. The systems which we will use were developed long before logic programming came into being. Among them, intuitionistic logic and modal logics are of special importance: it has been argued by philosophers and researchers in artificial intelligence that common sense reasoning employs methods of such systems.

In this paper we will analyze negation as failure (abbreviated as NF). NF is the central notion in the area of negation in logic programming. It serves as a model on which the variant implemented in Prolog is based, and it is a point of reference with which other versions of negation are compared. For an overview of issues related to negation the reader is referred to a survey paper by Apt and Bol.[1] NF is based on the following idea of Clark: In order to answer whether $\sim A$ is provable, try proving A; if the proof succeeds, answer *NO* to the original question; if the proof fails, answer *YES* to the original question.[6] Unfortunately the simplicity and elegance of this idea gets obscured by technical details if one considers programs which contain variables. For this reason in this paper we will investigate only programs without variables i.e. propositional programs. We believe that a great deal can be learned about the nature of general NF from considering this simpler case. Also, this class of programs is not without importance for applications: in rule-based expert systems knowledge is often represented in propositional logic.

We will be looking for mathematical results which could assure that programming can be declarative. According to the idea of declarative programming, one could write logic programs thinking of them as of specifications and not considering their procedural behavior under SLDNF-resolution:

> answers computed by SLDNF-resolution should be exactly the logical consequences of an easily understandable completion of the program.

This idea was first formalized by Clark.[6] His results can be reformulated as follows:

> SLDNF-resolution with *definite* program P and *positive* literal L as a query returns the answer *YES* iff $Comp(P) \vdash L$.
> SLDNF-resolution with *definite* program P and *positive* literal L as a query returns the answer *NO* iff $Comp(P) \vdash \neg L$.

The implications from the left to the right are called *soundness*; the implications from the right to the left are called *completeness*. Definite programs, for which the theorems above hold, do not admit negative literals in clauses' bodies. When negation is allowed in clauses' bodies Clark obtained only soundness:

> If SLDNF-resolution with *normal* program P and literal L as a query returns the answer *YES* then $Comp(P) \vdash L$.
> If SLDNF-resolution with *normal* program P and literal L as a query returns the answer *NO* then $Comp(P) \vdash \neg L$.

It is well known that the converse implications, i.e. completeness, do not hold. This is demonstrated by the following example.

Example 1 Consider the following normal program P:

$$\begin{aligned} &p \leftarrow q \\ &p \leftarrow \neg q \\ &q \leftarrow q \end{aligned}$$

Given P and query p SLDNF-resolution loops infinitely. On the other hand Clark's completion $Comp(P)$ of this program is:

$$\begin{aligned} &p \equiv q \vee \neg q \\ &q \equiv q \end{aligned}$$

and we have $Comp(P) \vdash p$. □

In our approach we will define and investigate the notion of constructive transformation of a program. We will show that this notion can replace negation as failure, can be used to characterize the SLDNF-computations and can eliminate the need for multivalued semantics.

In this paper our intention is to present the material in a way which allows us to rethink basic properties of SLDNF-resolution and notice symmetries of this notion which have been overlooked or which do not receive sufficient emphasis in other literature.

Here is the outline of the paper. Section 2 defines the terminology and notation to be used in the paper and explores basic symmetries of SLDNF-resolution. Section 3 presents the constructive transformation P^+ of a program P and shows how it can eliminate the use of negation as failure. It also contains a characterization of SLDNF-resolution using interpretations of logic programs in classical, intuitionistic and intermediate logics. One of the subsections will recall the syntax and semantics of modal logics and give yet another characterization of SLDNF-resolution. Section 4 compares the two valued fix-point and declarative semantics for P^+ with multivalued semantics for the original program P. Section 5 concludes the paper.

2. Basic Properties of Propositional SLDNF-resolution

2.1. Basic Definitions

We assume the terminology, definitions and basic results from chapters 1-3 of *Foundations of Logic Programming* by J.W. Lloyd, but some additions will be needed.[19]

We will generalize the notion of a program by admitting arbitrary formulas with $\top, \bot, \wedge, \vee, \neg$ in the bodies of statements. With this generalization every program can be transformed to Clark's form, i.e. to the form in which every propositional symbol occurs as the head of exactly one statement. Having programs in Clark's form will be essential for further considerations. We will extend SLDNF-resolution to handle such generalized programs and notice that all the basic definitions and results of semantics extend to this setting.

We will also define a number of straightforward but useful transformations of programs. Then, as the notion of a finite failure set is defined in Lloyd's book only for definite programs, according to our needs, we will extend this notion to programs containing negations in statement bodies.

Finally, we will be considering SLDNF-computations, i.e. sets of possible derivations. Recall that SLDNF is an idealized version of Prolog. At the propositional level there are only two differences between Prolog and SLDNF-resolution. In Prolog atoms in a goal are selected in the left-to-right order while in SLDNF selection is nondeterministic, and Prolog selects rules in derivations in a systematic manner using backtracking, while SLDNF selects rules nondeterministically. We will define notation $\mathit{SLDNF}(P,L)=\mathit{YES}$, $\mathit{SLDNF}(P,L)=\mathit{NO}$ and $\mathit{SLDNF}(P,L)=\uparrow$, which intuitively correspond to situations when the idealized Prolog answers YES, NO, or loops infinitely. This notation will be later used in several non-procedural characterizations of SLDNF-computations.

Now the details. We write $\top, \bot$ to denote the *truth constants*: true and false. We write $\leftarrow$ to denote reversed implication: $A \leftarrow B$ is the same as $B \rightarrow A$. The formula $\leftarrow B$ is identified with $\bot \leftarrow B$, which is equivalent to $\neg B$. The empty conjunction is identified with $\top$; the empty disjunction is identified with $\bot$.

Definition 2 Throughout this paper symbol *Prop* will represent a set of propositional symbols such that $\mathit{Prop} \cap \{\bar{p} \mid p \in \mathit{Prop}\} = \emptyset$. □

By a *propositional atom* we understand any propositional symbol; truth constants are not atoms. By a *propositional literal* we understand either a propositional symbol or a negation of a propositional symbol. By a *propositional definite clause* we understand any formula $A \leftarrow B$ where A is an atom and B is a conjunction of atoms and truth constants. By a *propositional normal clause* we understand any formula $A \leftarrow B$ where A is an atom and B is a conjunction of literals and truth constants. The next definition generalizes these notions.

Definition 3

By a *propositional positive formula* we understand any formula built from propositional atoms and truth constants by means of $\wedge, \vee$. By a *propositional positive statement* we understand any formula $A \leftarrow B$ where A is a propositional atom and B is a propositional positive formula. By a *propositional positive program* we understand any finite set of propositional positive statements. By a *propositional positive goal* we understand any formula $\leftarrow B$ where B is a propositional positive formula.

By a *propositional statement* we understand any formula $A \leftarrow B$ where A is a propositional atom and B is an arbitrary propositional formula built from propositional atoms and truth constants by means of $\wedge, \vee, \neg$. By a *propositional general program* we understand any finite set of propositional statements. By a *propositional general goal* we understand any propositional formula $\leftarrow B$ where B is arbitrary propositional formula built from propositional atoms and truth constants by means of $\wedge, \vee, \neg$. $\square$

Definition 4

1. A propositional general program P over $Prop$ is said to be in *Clark's form* if for every $p \in Prop$, P contains exactly one statement defining p.

2. We define *Clark's transformation* $(\cdot)^C$ which transforms a propositional general program P over $Prop$ into propositional general program P^C as follows. For every propositional symbol $p \in Prop$ list all the statements defining p: $p \leftarrow B_1$, ... , $p \leftarrow B_n$, and form $p \leftarrow \bigvee_{i \leq n} B_i$. If there are no statements defining p, form $p \leftarrow \bot$. The program P^C consists of all statements obtained in this way.

3. *Clark's completion* $Comp(P)$ of the propositional general program P is defined as $Comp(P) = \{(p \equiv B) \mid (p \leftarrow B) \in P^C\}$. $\square$

Notice that in order to apply Clark's transformation it is important to know the set of propositional symbols intended for the program. While applying Clark's transformation, for every propositional symbol which does not occur in the program, we need to add a clause $p \leftarrow \bot$. The definition of Clark's transformation is the only part of our considerations which requires the assumption that P is finite.

The standard definition of SLDNF-resolution covers only normal programs and normal goals. We will extend it to general programs and goals through a series of transformations.

Definition 5

1. We define *deMorgan transformation* $(\cdot)^{deM}$ which can be applied to formulas built from atoms and truth constants by means of $\wedge, \vee, \neg$, and to statements and general programs.
   $\top^{deM} = \top$,
   $\bot^{deM} = \bot$,
   $\neg\top^{deM} = \bot$,
   $\neg\bot^{deM} = \top$,
   $p^{deM} = p$ for any propositional symbol p,
   $(\neg p)^{deM} = \neg p$ for any propositional symbol p,
   $(\neg\neg B)^{deM} = B$,
   $(B_1 \wedge B_2)^{deM} = B_1^{deM} \wedge B_2^{deM}$,
   $(\neg(B_1 \wedge B_2))^{deM} = (\neg B_1)^{deM} \vee (\neg B_2)^{deM}$,
   $(B_1 \vee B_2)^{deM} = B_1^{deM} \vee B_2^{deM}$,
   $(\neg(B_1 \vee B_2))^{deM} = (\neg B_1)^{deM} \wedge (\neg B_2)^{deM}$,
   $(A \leftarrow B)^{deM} = A \leftarrow B^{deM}$ for any statement $(A \leftarrow B)$,
   $P^{deM} = \{(A \leftarrow B)^{deM} \mid (A \leftarrow B) \in P\}$ for any general program P.

2. Let us choose arbitrarily, and fix for our considerations, a deterministic algorithm which takes a formula built from literals and truth constants by means of $\wedge, \vee$ and returns an equivalent formula in disjunctive normal form, such that the algorithm does not use any transformations of subformulas except[a]:
   replace $B_1 \wedge (B_2 \vee B_3)$ by $(B_1 \wedge B_2) \vee (B_1 \wedge B_3)$ and
   replace $(B_1 \vee B_2) \wedge B_3$ by $(B_1 \wedge B_3) \vee (B_2 \wedge B_3)$.
   If B is the input formula, denote the output of this algorithm by $dnf(B)$.
   We define *dnf transformation* $(\cdot)^{dnf}$.
   $B^{dnf} = dnf(B^{deM})$, for any formula B built from atoms and $\top, \bot, \wedge, \vee, \neg$,
   $(A \leftarrow B)^{dnf} = A \leftarrow (B^{deM})^{dnf}$ for any statement $A \leftarrow B$,
   $P^{dnf} = \{(A \leftarrow B)^{dnf} \mid (A \leftarrow B) \in P\}$ for any general program P.

3. We define *separation transformation* $(\cdot)^S$ which transforms a propositional general program and goal $P; \leftarrow B$ into a propositional normal program P^S and a collection of propositional normal goals $(\leftarrow B)^S$. To obtain P^S, for every statement $A \leftarrow \bigvee_{i\in I} \bigwedge_{j\in J_i} L_{i,j}$ of P^{dnf} put into P^S the collection of clauses $A \leftarrow \bigwedge_{j\in J_i} L_{i,j} \;:\; i \in I$. For a propositional general goal $\leftarrow B$, if B^{dnf} is $\bigvee_{i\in I} \bigwedge_{j\in J_i} L_{i,j}$ then $(\leftarrow B)^S$ is the collection of normal goals
   $\leftarrow \bigwedge_{j\in J_i} L_{i,j} \;:\; i \in I$.

4. We say that there exists an *SLDNF-refutation for a propositional general program* $P; \leftarrow B$ if there exists $(\leftarrow B_j) \in (\leftarrow B)^S$ and there exists an SLDNF-refutation for the normal program and goal $P^S; \leftarrow B_j$.
   We say that there exists an *SLDNF-finite failure tree for propositional general program* $P; \leftarrow B$ if for every $(\leftarrow B_j) \in (\leftarrow B)^S$ there exists an SLDNF-finite failure tree for the normal program and goal $P; \leftarrow B_j$.

[a] From this assumption it follows that the algorithm does not replace $p \vee \neg p$ by $\top$ and does not replace $p \wedge \neg p$ by $\bot$.

SLD-resolution is generalized to positive programs and goals in an analogous way. □

The standard definition of the finite failure set concerns only definite programs.[19] In the next definition we extend this notion to general programs.

Definition 6 Let P be a propositional general program over *Prop*. By the *SLDNF-finite failure set* for P we understand the set of all propositional symbols $p \in \mathit{Prop}$, such that there exists a finitely failed SLDNF-tree for $P; \leftarrow p$. □

For a definite programs, the standard notion of SLD-finite failure set coincides with the notion of SLDNF-finite failure set above.[19]

One of the problems in the theory of logic programming is finding a non-procedural characterization of existence/non-existence of the successful or failed SLDNF-derivations. The next definition introduces notation convenient for discussing this problem.

Definition 7 Let $P; \leftarrow B$ be a propositional general program and goal.

1. $\mathit{SLDNF}(P, B) = \mathit{YES}$ is read:
   "SLDNF-computation for $P; \leftarrow B$ returns answer YES" and means:
   there exists an SLDNF-refutation for $P; \leftarrow B$.

2. $\mathit{SLDNF}(P, B) = \mathit{NO}$ is read:
   "SLDNF-computation for $P; \leftarrow B$ returns answer NO" and means:
   there exists a finitely failed SLDNF-tree for $P; \leftarrow B$.

3. $\mathit{SLDNF}(P, B) = \uparrow$ is read:
   "SLDNF-computation for $P; \leftarrow B$ loops infinitely" and means:
   there exists neither an SLDNF-refutation nor a finitely failed tree for $P; \leftarrow B$.

For propositional positive programs and goals under SLD-resolution, the notation $\mathit{SLD}(P, B)$=*YES*, $\mathit{SLD}(P, B)$=*NO*, $\mathit{SLD}(P, B)=\uparrow$ is defined analogously. □

Basic properties of transformations of programs are stated without proof in the following proposition.

Proposition 8 *Let $P; \leftarrow B$ be a propositional general program and goal. Then:*

1. *Every two of the following are equivalent in classical logic: B, B^{deM}, B^{dnf}.*

2. *Every two of the following are equivalent in classical logic: P, P^C, P^{deM}, P^{dnf}, P^S.*

3. *Every two of the following are equivalent in classical logic: $\mathit{Comp}(P)$, $\mathit{Comp}(P^C)$, $\mathit{Comp}(P^{deM})$, $\mathit{Comp}(P^{dnf})$, $\mathit{Comp}(P^S)$.*

4. *Each of the following returns the same result: $\mathit{SLDNF}(P, B)$, $\mathit{SLDNF}(P^C, B)$, $\mathit{SLDNF}(P^{deM}, B^{deM})$, $\mathit{SLDNF}(P^{dnf}, B^{dnf})$, $\mathit{SLDNF}(P^S, B^{dnf})$.*

Because of this proposition, when we work in classical logic, we will be allowed to use freely any of the transformations defined above.

2.2. Basic Properties of Propositional SLDNF-resolution

First let us state the relation of the notion of SLDNF-computation and success and finite failure sets. The following proposition results immediately from the definitions of the SLDNF- success set and SLDNF-finite failure set.

Proposition 9 *Let P be a propositional general program and let p be a propositional symbol. Then:*

1. p belongs to the SLDNF-success set of P	*iff*	$SLDNF(P, p) = YES$,
2. p belongs to the SLDNF-finite failure set of P	*iff*	$SLDNF(P, p) = NO$.

The next two Propositions show internal symmetries of the notion of SLDNF-computation.

Proposition 10 *Let $P; \leftarrow B$ be a propositional general program and goal. Then exactly one of the following conditions holds:*

$$SLDNF(P, B) = YES, \quad SLDNF(P, B) = NO, \quad SLDNF(P, B) =\uparrow.$$

Notice that an analogous property holds for SLD-resolution and propositional positive programs.

Proof. It is clear that at least one of these conditions holds. Indeed, by definition $SLDNF(P, B) =\uparrow$ if neither $SLDNF(P, B) = YES$ nor $SLDNF(P, B) = NO$. For the same reason we can see that conditions $SLDNF(P, B) = YES$ and $SLDNF(P, B) =\uparrow$ are disjoint, as well as $SLDNF(P, B) = NO$ and $SLDNF(P, B) =\uparrow$ are disjoint. It remains to show that $SLDNF(P, B) = YES$ and $SLDNF(P, B) = NO$ are disjoint. Restricting Theorem 9.2 (p. 51) in Lloyd's book to the propositional case and allowing general programs, we obtain: If there is a successful derivation for $P; \leftarrow B$, then for every computation rule R there is a successful derivation for $P; \leftarrow B$ via R.[19] So, if there is a successful derivation for $P; \leftarrow B$, then every SLDNF-tree for $P; \leftarrow B$ contains a successful branch, i.e. there are no finite failure trees. □

Proposition 11 *Let $P; \leftarrow B$ be a propositional general program and goal. Then:*

1. $SLDNF(P, B) = YES$	*iff*	$SLDNF(P, \neg B) = NO$,
2. $SLDNF(P, B) = NO$	*iff*	$SLDNF(P, \neg B) = YES$,
3. $SLDNF(P, B) = \uparrow$	*iff*	$SLDNF(P, \neg B) = \uparrow$.

Proof. Assume that disjunctive normal form of B is $\bigvee_{i \in I} B_i$ and disjunctive normal form of $\neg B$ is $\bigvee_{j \in J} B'_j$. Notice that for any B_i, every B'_j must contain a negation of a literal from B_i. Also, notice that if you select one literal from every B'_j, then there exists a B_i such that the negations of its literals are contained among the literals you selected. Now we can prove the first equivalence. Below, a negation of a negative literal $\neg p$ will be identified with p.

$SLDNF(P, B) = YES$, iff
there is $i \in I$ such that $SLDNF(P, B_i) = YES$, iff
there is $i \in I$ such that for every conjunct L in B_i, $SLDNF(P, L) = YES$, iff
there is $i \in I$ such that for every conjunct L in B_i, $SLDNF(P, \neg L) = NO$, iff
for every $j \in J$ there is a literal $\neg L$ in B_j such that $SLDNF(P, \neg L) = NO$, iff
for every $j \in J$, $SLDNF(P, B_j) = NO$, iff

$SLDNF(P, \neg B) = NO$.
The second equivalence in the proposition is a reformulation of the first one. The third equivalence follows from the first and the second by Proposition 10. $\square$

SLDNF-resolution for propositional general programs and goals can be characterized by means of a game in the style of Ehrenfeucht.

Remark 12 Given a propositional general program P and goal B we define game $GAME(P, B)$. Two players participate in this game. Each player assumes either a role of a *proponent* or a role of an *opponent*; their roles may change during the game. At every point of the game they consider a formula and replace it by another formula. Who moves next depends on the current formula. The game starts with a Player 1 being a proponent, Player 2 being an opponent, and initial formula being B. If the current formula is a disjunction $B_1 \vee \ldots \vee B_n$, the proponent replaces it by one of the disjuncts B_i. If the current formula is a conjunction $B_1 \wedge \ldots \wedge B_n$, the opponent replaces it by one of the conjuncts B_i. If the current formula is a negation $\neg B$, it is replaced by B and the players change roles: the proponent becomes an opponent and the opponent becomes a proponent. If the current formula is a propositional symbol p, the proponent chooses a statement $p \leftarrow B'$ from P and replaces the current formula by B'; if there is no such a statement, the current formula is replaced by $\perp$. If the current formula is $\top$ the game ends, the proponent being a winner. If the current formula is $\perp$ the game ends, the opponent being a winner.

One can prove that:
$SLDNF(P, B) = YES$ iff Player 1 has a winning strategy in $GAME(P, B)$,
$SLDNF(P, B) = NO$ iff Player 2 has a winning strategy in $GAME(P, B)$,
$SLDNF(P, B) = \uparrow$ iff neither player has a winning strategy in $GAME(P, B)$. $\square$

3. Constructive Transformation and Characterizations of Propositional SLDNF-resolution

3.1. Constructive Transformation and Characterization of Propositional SLDNF-resolution in Classical Logic

"I learned to recognize the thorough and primitive duality of man; I saw that, of the two natures that contended in the field of my consciousness, even if I could rightly be said to be either, it was only because I was radically both; and from an early date, even before the course of my scientific discoveries had begun to suggest the most naked possibility of such a miracle, I had learned to dwell with pleasure, as a beloved day-dream, on the thought of the separation of these elements. If each, I told myself, could but be housed in separate identities, life would be relieved of all that was unbearable [...] It was the curse of mankind that these incongruous faggots were thus bound together – that in the agonized womb of consciousness, these polar twins should be continuously struggling. How, then, were they dissociated?"

Robert Louis Stevenson, *Strange Case of Dr. Jekyll and Mr. Hyde*, 1886.

We will define the constructive completion $Comp(P^+)$ for a propositional general program P, and prove soundness and completeness of propositional SLDNF-resolution. $Comp(P^+)$ is different from Clark's completion $Comp(P)$ – it uses an extended language in which every propositional symbol p has a counterpart $\overline{p}$. Informally speaking, $\overline{p}$ represents the negation of p in a positive way. This idea originates from the Prolog programming method of constructive negation: instead of using a negative literal $\neg A$, define in a positive way a new predicate with the same properties as $\neg A$. In this way positive and negative aspects of a predicate can be dissociated.

Definition 13 Let *Prop* be a set of propositional symbols such that $Prop \cap \{\bar{p} \mid p \in Prop\} = \emptyset$. We define $Prop^{\#} = Prop \cup \{\bar{p} \mid p \in Prop\}$. □

Definition 14 Let *Form* be the set of all formulas over a given set of propositional symbols built using connectives $\top, \bot, \wedge, \vee, \neg$. We define *positive transformation* $(\cdot)^+$ and *negative transformation* $(\cdot)^-$:

$$\begin{array}{ll}
(\cdot)^+ : Form(Prop) \rightarrow Form(Prop^{\#}) & (\cdot)^- : Form(Prop) \rightarrow Form(Prop^{\#}) \\
p^+ = p & p^- = \bar{p} \\
\top^+ = \top & \top^- = \bot \\
\bot^+ = \bot & \bot^- = \top \\
(B_1 \wedge B_2)^+ = B_1^+ \wedge B_2^+ & (B_1 \wedge B_2)^- = B_1^- \vee B_2^- \\
(B_1 \vee B_2)^+ = B_1^+ \vee B_2^+ & (B_1 \vee B_2)^- = B_1^- \wedge B_2^- \\
(\neg B)^+ = B^- & (\neg B)^- = B^+
\end{array}$$
□

Definition 15 (Constructive transformation and completion)

1. Let P be a propositional general program over *Prop*. By a *constructive transformation* of P we understand a propositional general program over $Prop^{\#}$ obtained as
$$P^+ = \{A^+ \leftarrow B^+ \mid (A \leftarrow B) \in P^C\} \cup \{A^- \leftarrow B^- \mid (A \leftarrow B) \in P^C\}.$$

2. *Constructive transformation* of a propositional general goal $\leftarrow B$ is $\leftarrow B^+$.

3. *Constructive completion* of propositional general program P is defined as $Comp(P^+)$. □

Example 16 Let P be the following propositional general program:

$$\begin{array}{l} p \leftarrow p \\ q \leftarrow \neg q \\ q \leftarrow p \end{array}$$

We will produce the constructive completion $Comp(P^+)$. First we turn P into Clark's form P^C:

$$\begin{array}{l} p \leftarrow p \\ q \leftarrow \neg q \vee p \end{array}$$

The constructive transformation P^+ is:

$$\begin{array}{ll} p \leftarrow p & \overline{p} \leftarrow \overline{p} \\ q \leftarrow \overline{q} \vee p & \overline{q} \leftarrow \ q \wedge \overline{p} \end{array}$$

Now, $Comp(P^+)$ is obtained by changing implications to equivalences:

$$\begin{array}{ll} p \equiv p & \overline{p} \equiv \overline{p} \\ q \equiv \overline{q} \vee p & \overline{q} \equiv \ q \wedge \overline{p} \end{array}$$
□

Notice that the constructive transformation P^+ is always a positive program.

Notice that $Comp(P)$ may be inconsistent; this is for instance the case for P containing just one clause: $p \leftarrow \neg p$. Thus the following proposition shows one of the important differences between $Comp(P)$ and $Comp(P^+)$.

Proposition 17 *Let P be a propositional general program. Then the constructive completion $Comp(P^+)$ is consistent in classical logic.*

Proof. P^+ is a positive program. It is equivalent to $(P^+)^S$ which is a definite program. The least Herbrand model for $(P^+)^S$ is a model for $Comp(P^+)$. □

The constructive transformation P^+ is in some sense symmetrical. A precise description of its internal symmetries is given in the next proposition.

Proposition 18 *Let $P; \leftarrow B$ be a propositional general program and goal. Then:*

1. $SLD(P^+, B^+) = YES$ *iff* $SLD(P^+, (\neg B)^+) = NO$.
2. $SLD(P^+, B^+) = NO$ *iff* $SLD(P^+, (\neg B)^+) = YES$.
3. $SLD(P^+, B^+) = \uparrow$ *iff* $SLD(P^+, (\neg B)^+) = \uparrow$.

The reader should compare this proposition with Proposition 11. The proof will be given in the next section.

The next theorem shows that for propositional general programs SLDNF-resolution can be replaced by SLD-resolution and constructive transformation.

Theorem 19 *Let $P; \leftarrow B$ be a propositional general program and goal. Then:*

1. $SLDNF(P, B) = YES$ *iff* $SLD(P^+, B^+) = YES$.
2. $SLDNF(P, B) = NO$ *iff* $SLD(P^+, B^+) = NO$.
3. $SLDNF(P, B) = \uparrow$ *iff* $SLD(P^+, B^+) = \uparrow$.

The proof will be given in the next section.

This theorem suggests the following important strategy for handling programs with negation: instead of performing SLDNF resolution on $P; \leftarrow B$, construct the positive program and goal $P^+; \leftarrow B^+$ and perform SLD-resolution. This idea can be carried over to the case of first-order programs with $\top, \bot, \wedge, \vee, \neg, \forall, \exists$ and a resolution system which extends SLDNF and is capable of handling such programs.[22,24]

The following theorem shows that the constructive completion fully characterizes the behavior of propositional SLDNF-resolution.

Theorem 20 (Soundness and completeness)
Let $P; \leftarrow B$ be a propositional general program and goal. Then:

1. $SLDNF(P, B) = YES$ *iff* $Comp(P^+) \vdash B^+$.
2. $SLDNF(P, B) = NO$ *iff* $Comp(P^+) \vdash (\neg B)^+$.
3. $SLDNF(P, B) = \uparrow$ *iff* $Comp(P^+) \not\vdash B^+$ *and* $Comp(P^+) \not\vdash (\neg B)^+$.

Remark 21 It is well known that for a definite program P',

$$P' \vdash p \text{ iff } Comp(P') \vdash p,$$

and the same holds for any positive program. Because of this equivalence, in the last theorem we could use P^+ instead of $Comp(P^+)$. □

Theorem 20 results from Theorem 19 and Clark's theorem on soundness and completeness of SLD-resolution. The details of the proof will be given in the next section.

Example 22 Continuing Example 16, notice that for the same program, $SLDNF(P, p) =\uparrow$. At the same time p follows from Clark's completion: $Comp(P)$

is $\{p \equiv p, q \equiv (\neg q \vee p)\}$ and we can see that $Comp(P) \vdash p$. So this example shows that SLDNF-resolution is not complete with respect to $Comp(P)$. Let us now see how the constructive completion $Comp(P^+)$ behaves in this situation. As stated before, $Comp(P^+)$ contains: $p \equiv p$, $\overline{p} \equiv \overline{p}$, $q \equiv \overline{q} \vee p$, $\overline{q} \equiv q \wedge \overline{p}$. We can check that $Comp(P^+) \not\vdash p$ and $Comp(P^+) \not\vdash \overline{p}$ so according to Theorem 20, $SLDNF(P, p) = \uparrow$. □

Now let us discuss the notion of $Comp(P^+)$, thanks to which we achieved completeness. Example 22 shows that Clark's completion is in some sense too strong, it implies things that are not computed by resolution. The constructive completion $Comp(P^+)$ is weaker; roughly speaking:

$$Comp(P) = Comp(P^+) + \{\neg p \equiv \overline{p} \mid p \in Prop\}.$$

The precise relation between the two notions of completion is given in the following proposition.

Proposition 23 *Let P be a propositional general program over Prop. Then (in classical logic):*

1. *$Comp(P) \cup \{\neg p \equiv \overline{p} \mid p \in Prop\}$ is equivalent to $Comp(P^+) \cup \{\neg p \equiv \overline{p} \mid p \in Prop\}$.*
2. *$Comp(P) \cup \{\neg p \equiv \overline{p} \mid p \in Prop\}$ is a conservative extension of $Comp(P)$.*

Proof.

1. Straightforward.
2. As this extension is definitional, it is conservative. □

Constructive transformation defined in this section is closely related to intensional negation.

Remark 24 Intensional negation was defined in P. Mancarella's doctoral dissertation and subsequent papers.[3,4] The notion initially was applied to definite programs, to which possibly negative queries could be asked. In a paper in the 1990 North American Logic Programming Conference Mancarella, Pedreschi, Rondinelli and Tagliatti, defined an algebra of logic programs.[20] The elements of this algebra are normal programs over a fixed alphabet, with certain restrictions on occurrences of variables. The algebra is equipped with operations $\cup, \cap, Not$, which are based on semantical ideas, with *Not* being an extension of the intensional negation. (We suppose that this algebra is a Boolean algebra and believe it is worth further investigation.) Using the operations of the algebra, the authors define a transformation of a program P into a new program denoted $P^- \cup P^+$. For the purpose of discussion we will change their notation to $P^\ominus \cup P^\oplus$. One can prove that for a propositional normal program P, the following relationship holds between $P^\ominus \cup P^\oplus$ and our P^+:

$$P^+ = P^\ominus \cup P^\oplus.$$

In the light of this equality, Propositions 4 and 6 stated without proofs in the paper of Mancarella et al. are equivalent to our Theorem 19 and Lemma 28. □

Remark 25 Drabent and Martelli defined the following operation.[8] Given a normal program P, add to P its copy P' in which every predicate symbol p is replaced by a new symbol p'; then replace every $\neg p$ in P by $\neg p'$ and every $\neg p'$ in P' by $\neg p$. The obtained program is denoted $split(P)$. In Theorem 6 the authors proved that for an allowed program and goal $P; \leftarrow B$ the following equivalence holds.

$$Comp(split(P)) \models split(B)\theta \ \text{ iff } \ SLDNF(P; \leftarrow B) \text{ computes } \theta \qquad (*)$$

The proof relies on a reduction of this equivalence to Kunen's completeness result in three-valued logic. Let us observe that for a propositional normal program P over $Prop$ the following relation holds between $split(P)$ and P^+:

$$\{\neg p' \equiv \overline{p} \mid p \in Prop\} \models split(P) \equiv P^+$$

In the light of this equivalence, $(*)$ is a generalization of our Theorem 20. Later in our paper we will consider characterizations of SLDNF-resolution in intuitionistic and intermediate logics; it would be interesting to investigate whether for allowed programs the equivalence $(*)$ generalizes to such a setting. □

Let us also mention that *doubled program* and *partial completion* are yet another reincarnation of constructive completion.[28,29] As the research for which these notions were used goes in a different direction than ours, we will not discuss them in detail.

3.2. Proofs

In this section we will prove results related to soundness and completeness which were stated in Subsection 3.1.

The idea of the proof is the following: First we prove the following equivalences:

(i) $SLDNF(P, B) = YES$ iff $SLD(P^+, B^+) = YES$.

(ii) $SLDNF(P, B) = NO$ iff $SLD(P^+, B^+) = NO$.

Implications to the right in (i) and (ii) are proved in parallel by induction on the length of a successful SLDNF-derivation or on depth of SLDNF-finite failure tree. Implications to the left in (i) and (ii) require transforming SLD-derivations to a certain normal form and using inductions similar to the previous ones. Then by soundness and completeness of SLD-resolution for definite programs we obtain points 1 and 2 of Theorem 20. Point 3 follows from points 1 and 2 by Proposition 10.

Inductions used in the proofs in this section are straightforward but a little strenuous. The reader may wish to skip to the next subsection and come back here after having read the rest of the paper.

In the proofs we will use the following conventions.

If L is a negative literal $\neg A$, then $\neg L$ is identified with A.

If L is $\bar{q}$, $\bar{L}$ is identified with q.

If L is $\neg q$, then $\langle L \rangle$ is $\bar{q}$; if L is q, then $\langle L \rangle$ is q.

Lemma 26 *Let P be a propositional general program, and let p be a propositional symbol. Then:*

1. $SLD(P^+, p) = YES$ *implies* $SLD(P^+, \overline{p}) = NO$,
2. $SLD(P^+, \overline{p}) = YES$ *implies* $SLD(P^+, p) = NO$.

Proof. We prove both statements in parallel by induction on the length of successful SLD-derivation for $P^+; \leftarrow p$ or $P^+; \leftarrow \overline{p}$.

$SLD(P^+, p) = YES$ derived in 1 step.

Then the (unique) statement defining p in P^+ is of the form $p \leftarrow \top \vee B$, where B is a disjunction of conjunctions of atoms. Thus the (unique) statement defining $\overline{p}$ in P^+ is of the form $\overline{p} \leftarrow \bigvee_{i\in I}(\bot \wedge B_i)$, where each B_i is a conjunction of atoms. This is interpreted as a collection $\overline{p} \leftarrow \bot \wedge B_i \quad : i \in I$ of statements in a definite program. If we start from $\leftarrow \overline{p}$, and use each of these rules, and select atom $\bot$ in every derived goal, we obtain a finite failure tree for $P^+; \leftarrow \overline{p}$. So $SLD(P^+, \overline{p}) = NO$.

$SLD(P^+, \overline{p}) = YES$ derived in 1 step.
Similar to the case above.

$SLD(P^+, p) = YES$ derived in $n+1$ steps.
Let $p \leftarrow L_1 \wedge \ldots \wedge L_k$ be the clause in (the normal version of) P^+, with which the successful derivation for $P^+; \leftarrow p$ starts. By induction hypothesis, for each $L_i : i \leq k$ there is a finitely failed SLD-tree for $P; \leftarrow \overline{L}$. (If L is $\overline{q}$, by $\overline{L}$ we understand just q). The (unique) statement defining p in P^+ is of the form $p \leftarrow (L_1 \wedge \ldots \wedge L_k) \vee B$ where B is a disjunction of conjunctions of atoms. Thus every clause defining $\overline{p}$ in the normal version of P^+ contains one of $\overline{L}_i : i \leq k$ as a conjunct in its body: $p \leftarrow \ldots \wedge \overline{L}_i \wedge \ldots$. If we start from $\leftarrow \overline{p}$ and use each of these clauses, selecting in each of derived goals the corresponding atom $\overline{L}_i$ and constructing further finite failure trees for these atoms, we will obtain a finitely failed SLD-tree for $P; \leftarrow \overline{p}$. So $SLD(P^+, \overline{p}) = NO$.

$SLD(P^+, \overline{p}) = YES$ derived in $n+1$ steps.
Similar to the case above. □

Lemma 27 *Let P be a propositional general program, and let p be a propositional symbol. Then:*

1. $SLD(P^+, p) = NO$ *implies* $SLD(P^+, \overline{p}) = YES$,
2. $SLD(P^+, \overline{p}) = NO$ *implies* $SLD(P^+, p) = YES$.

Proof. We prove both statements in parallel by induction on the depth of finitely failed SLD-tree for $P^+; \leftarrow p$ or $P^+; \leftarrow \overline{p}$.

Finitely failed SLD-tree of depth 1 for $P; \leftarrow p$.
Then the (unique) statement defining p in P^+ is of the form $p \leftarrow \bigvee_i(\bot \wedge B_i)$, where each B_i is a conjunction of of atoms. Thus the unique statement defining $\overline{p}$ in P^+ is of the form $\overline{p} \leftarrow \top \vee B'$, where each B' is a disjunction of conjunctions of atoms. Using the clause $\overline{p} \leftarrow \top$ from the definite version of P^+, we obtain a successful derivation for $P; \leftarrow \overline{p}$.

Finitely failed SLD-tree of depth 1 for $P; \leftarrow \overline{p}$.
Similar to the case above.

Finitely failed SLD-tree of depth $n+1$ for $P; \leftarrow p$.
Let $p \leftarrow \ldots \wedge L_1 \wedge \ldots, \quad \ldots, \quad p \leftarrow \ldots \wedge L_k \wedge \ldots$ be the collection of all clauses in the definite version of P^+, with $L_1, \ldots, L_k$ being the literals selected in the construction of finitely failed SLD-tree of depth $n+1$ for $P; \leftarrow p$. By induction hypothesis, for each $i \leq k$ there is a successful derivation for $P; \leftarrow \overline{L}_i$. In the definite version of P^+ there has to be a clause $\overline{p} \leftarrow \overline{L}_1 \wedge \ldots \wedge \overline{L}_k$. Using this

clause, and then performing steps from successful derivations for $P; \leftarrow \overline{L_i}$ for each i, we obtain a successful derivation for $P^+; \overline{p}$. So $SLD(P, \overline{p}) = YES$.

Finitely failed SLD-tree of depth $n+1$ for $P; \leftarrow \overline{p}$.
Similar to the case above. □

Lemma 28 *Let P be a propositional general program, and let p be a propositional symbol. Then:*

1. $SLD(P^+, p) = YES$ *iff* $SLD(P^+, \overline{p}) = NO$,
2. $SLD(P^+, \overline{p}) = NO$ *iff* $SLD(P^+, p) = YES$,
3. $SLD(P^+, \overline{p}) = \uparrow$ *iff* $SLD(P^+, p) = \uparrow$.

Proof. 1 and 2 follow from Lemmas 26 and 27. Point 3 follows from 1 and 2 by Proposition 10. □

Now we can prove the result on the symmetry of SLD-resolution stated in the previous section.

Proof of Theorem 18.

1. By Lemma 28, if B is a literal, the equivalence 1 is true. Consider now arbitrary formula B, and assume that $\bigvee_i \bigwedge_j L_{i,j}$ is the disjunctive normal form of B. (In the following we will use notation defined at the beginning of this chapter.) We have:
   $SLD(P^+, B^+) = YES$, iff
   $SLD(P^+, \bigvee_i \bigwedge_j \langle L_{i,j} \rangle) = YES$, iff
   there is i such that $SLD(P^+, \bigwedge_j \langle L_{i,j} \rangle) = YES$, iff
   there is i such that for all j, $SLD(P^+, \langle L_{i,j} \rangle) = YES$, iff (by Lemma 28)
   there is i such that for all j, $SLD(P^+, \langle \neg L_{i,j} \rangle) = NO$, iff
   there is i such that $SLD(P^+, \bigvee_j \langle \neg L_{i,j} \rangle) = NO$, iff
   $SLD(P^+, \bigwedge_i \bigvee_j \langle \neg L_{i,j} \rangle) = NO$, iff
   $SLD(P^+, (\neg B)^+) = NO$.

2. Similar to 1.

3. From 1 and 2 by Proposition 10 □

Lemma 29 *Let P be a propositional general program, and let p be a propositional symbol. Then:*

1. $SLDNF(P, p) = YES$ *implies* $SLD(P^+, p) = YES$,
2. $SLDNF(P, p) = NO$ *implies* $SLD(P^+, p) = NO$.

Proof. We prove both statements in parallel by induction on the rank of the successful SLDNF-derivation for $P; \leftarrow p$, or on the rank of finitely failed tree for $P; \leftarrow p$.

Successful SLDNF-derivation of rank 0 for $P; \leftarrow p$.
The derivation is of rank 0, which means that no negative subgoals were selected. By replacing every negative literal $\neg q$ in the derivation by $\overline{q}$ we obtain a successful SLD-derivation for $P^+; p$. So $SLD(P^+, p) = YES$.

Finitely failed SLDNF-tree of rank 0 for $P; \leftarrow p$.

The tree is of rank 0, which means that no negative subgoals were selected. Replacing every negative literal $\neg q$ in the tree by $\overline{q}$ we obtain a finitely failed SLD-tree for $P^+; p$. So $SLD(P^+, p) = NO$.

Successful SLDNF-derivation of rank $n+1$ for $P; \leftarrow p$.

Consider each negative subgoal $\neg A$ selected in this derivation. $P; \leftarrow A$ has a finitely failed SLDNF-tree of rank $\leq n$, and by induction hypothesis, there exists a finitely failed SLD-tree for $P^+; \leftarrow A$, then by Lemma 27, there exists a successful SLD-derivation for $P^+; \leftarrow \overline{A}$. If we replace each selected negative subgoal $\neg A$ in the successful SLDNF-derivation for $P; \leftarrow p$ by the corresponding successful SLD-derivation for $P^+; \leftarrow \overline{A}$, and also each remaining negative literal $\neg q$ by $\overline{q}$, we will obtain a successful SLD-derivation for $P^+; \leftarrow p$. So $SLD(P^+, p) = YES$.

Finitely failed SLDNF-tree of rank $n+1$ for $P; \leftarrow p$.

Consider each $\neg A$ selected in each goal which is not the last goal on a branch of the tree. $P; \leftarrow \neg A$ has a successful SLDNF-derivation of rank $\leq n$, so by Proposition 11, $P; \leftarrow A$ has a finitely failed SLDNF-tree (and the tree constructed in the proof of that proposition is of rank $\leq n$. By induction hypothesis there is a finitely failed SLD-tree for $P^+; \leftarrow A$ and then by Lemma 28 there is a successful SLD-derivation for $P^+; \leftarrow \overline{A}$. Replace each such goal by the corresponding successful SLD-derivation for $P^+; \leftarrow \overline{A}$.

Consider each $\neg A$ selected in a goal which is the last goal on a branch of the tree. $P; \leftarrow A$ has a successful SLDNF-derivation of rank $\leq n$, and by induction hypothesis there is a successful SLD-derivation for $P^+; \leftarrow A$, then by Lemma 26 there exists a finitely failed SLD-tree for $P^+; \leftarrow \overline{A}$. Replace each such goal in the finitely failed SLDNF-tree for $P; \leftarrow A$ by the corresponding finitely failed SLD-tree for $P^+; \leftarrow \overline{A}$, then replace each remaining negative literal $\neg q$ in the tree by $\overline{q}$. We will obtain a finitely failed SLD-tree for $P^+; \leftarrow p$. So $SLD(P^+, p) = NO$. $\square$

The last proof shows that every SLDNF-derivation for general program P translates directly into an SLD-derivation for program P^+. The converse translation is more complex because not all SLD-derivations for P^+ correspond *directly* to SLDNF-derivations for P. Recall that in SLDNF-derivations negative subgoals are treated as "lemmas" which must be completed before other literals can be selected and resolved. On the other hand in an SLD-derivation for P^+, given a goal, say, $\leftarrow q_1 \wedge \overline{q}_2$, we could select atom $\overline{q}_2$, replace it by, say q_3 obtaining $\leftarrow q_1 \wedge q_3$ and then instead of selecting q_3 (to complete the "lemma"), we could select q_1, etc. As the "lemma" starting from $\overline{q}_2$ has not been completed, such a derivation does not translate directly into an SLDNF-derivation. To deal with this problem we define *systematic derivations* for P^+, as those in which "lemmas" are treated appropriately; we prove that it is enough to use such derivations, and that they translate directly into SLDNF-derivations.

Definition 30

1. Let $G: \leftarrow A_1 \wedge \ldots \wedge A_{k-1} \wedge A_k \wedge A_{k+1} \wedge \ldots A_m$
and $G': \leftarrow A_1 \wedge \ldots \wedge A_{k-1} \wedge A'_1 \wedge \ldots \wedge A'_n \wedge A_{k+1} \wedge \ldots A_m$
be two consecutive goals in an SLD-derivation.
Then atoms $A'_1, \ldots, A'_n$ in G are called *children* of A_k in G'.
Each A_i $(1 \leq i \leq m, i \neq k)$ in G' is called a *copy* of the corresponding atom A_i in G. A copy of a copy is also called a copy.
By a *descendant* we understand either a child, or a copy, or a descendant of a descendant. If A'' is a descendant of A', then A' is called an *ancestor* of A''.

2. An SLD-derivation is called *systematic* if for every goal which is not the last goal, one of the following holds:

   - the selected atom has children and in the next step one of its children is selected,
   - the selected atom is the only atom in the goal and it does not have any children (so the next goal is $\leftarrow \emptyset$),
   - the selected atom A does not have any children but there are other atoms in this goal; in the next step one of those atoms is selected, which is a copy of the closest ancestor of A.

3. A finitely failed SLD-tree is called *systematic* if its every branch is a systematic derivation. □

Remark 31

1. Every SLD-derivation via the computation rule "select first from the left" is systematic. (This is the rule that is used in Prolog).

2. It is known that: If $P; \leftarrow p$ has a successful SLD-derivation, then it has a successful SLD-derivation via the rule "select first from the left". Cf. Thm. 9.2, p. 51 in Lloyd's book.[19] The same is not true for finitely failed trees. Consider program P:
$$p \leftarrow q \wedge r$$
$$r \leftarrow \bot$$
$$q \leftarrow q$$
Then $P; \leftarrow p$ has a finitely failed SLD-tree, but the tree constructed using the rule "select first from the left" is not finite.

3. According to the standard definition, a *computation rule* in a derivation, is a function from a set of definite goals to a set of atoms, such that the value of the function for a goal in the derivation is the atom that gets selected.[19] With this definition, not every derivation is a derivation via a computation rule. Indeed, consider the following program P:
$$p \leftarrow \top$$
$$q \leftarrow p \wedge q$$

$$q \leftarrow \top$$

and the derivation (selected atoms are underlined):

$$\begin{array}{l} \leftarrow \underline{p} \wedge q \\ \leftarrow \underline{q} \\ \leftarrow p \wedge \underline{q} \\ \leftarrow \underline{p} \\ \leftarrow \emptyset \end{array}$$

□

Definition 32 Let P be a propositional general program. Let $\mathcal{D}$ be a systematic SLD-derivation for P^+.

1. By a *lemma* in $\mathcal{D}$ we understand a fragment of the derivation which starts with goal $G :\leftarrow A_1 \wedge \ldots \wedge A_{k-1} \wedge A_k \wedge A_{k+1} \wedge \ldots A_m$, and ends with $G' :\leftarrow A_1 \wedge \ldots \wedge A_{k-1} \wedge A_{k+1} \wedge \ldots A_m$, such that each A_i in G' is a descendant of the corresponding A_i in G.
2. If $\mathcal{D}$ is a failed derivation, by a *failed lemma* we understand any final fragment of $\mathcal{D}$ which starts with a certain goal G and does not contain any lemma starting with G.
3. By a *generalized lemma* we understand either a lemma or a failed lemma.
4. A generalized lemma is *positive* if it starts with a goal in which a "positive" atom q is selected. A generalized lemma is *negative* if it starts with a goal in which a "negative" atom $\overline{q}$ is selected.
5. Consider maximal sequences of nested generalized lemmas in $\mathcal{D}$. For each such sequence S, count the number n_S of changes in the sequence of signs of the lemmas. *Rank* of the derivation $\mathcal{D}$ is defined as: $\max_S(n_S)$.
6. By a *rank* of a systematic finitely failed SLD-tree we understand maximum of ranks of the derivations which are branches of the tree. □

Example 33 Consider the following program:

$$\begin{array}{l} p \leftarrow \top \\ q \leftarrow \bot \end{array}$$

and derivation:

$$\begin{array}{l} \leftarrow p \wedge q \wedge r \\ \leftarrow q \wedge r \\ \leftarrow \bot \wedge r \end{array}$$

This derivation is systematic. The part consisting of the first two goals is a positive lemma. The part consisting of the second and third goal is a positive failed lemma. The segment consisting of all the three goals is neither a lemma nor a failed lemma. Rank of the derivation is 0. □

Lemma 34 *Let P be a propositional general program and let p be a propositional symbol. Then:*

1. *If $P^+; \leftarrow p$ has a successful SLD-derivation, then it has a systematic one.*
2. *If $P^+; \leftarrow p$ has a finitely failed SLD-tree, then it has a systematic one.*

Proof.

1. By Theorem 9.2, p. 51 in Lloyd's book, every successful SLD-derivation can be replaced by a successful SLD-derivation via the rule "select first from the left".[19] Such a derivation is systematic.

2. (As mentioned in the last remark, the rule "select first from the left" is of no use for finitely failed trees—in our proof we have to use another idea.) Assume that $P^+; \leftarrow p$ has a finitely failed SLD-tree. Then by Lemma 27, $P^+; \leftarrow \overline{p}$ has a successful SLD-derivation $\mathcal{D}$ which by point 1 can be assumed to be systematic. Using the construction from the proof of Lemma 26, translate $\mathcal{D}$ into a finitely failed SLD-tree for $P^+; \leftarrow p$. The resulting tree is systematic. □

Let us remark that the lemma above holds not only for programs of the form P^+ but for arbitrary definite programs. Proving this requires however using another construction in point 2.

Lemma 35 *Let P be a propositional general program and let p be a propositional symbol. Then:*

1. *There exists a successful systematic SLD-derivation of rank n for $P^+; \leftarrow p$ iff there exists a systematic finitely failed SLD-tree of rank n for $P^+; \leftarrow \overline{p}$.*

2. *There exists a successful systematic SLD-derivation of rank n for $P^+; \leftarrow \overline{p}$ iff there exists a systematic finitely failed SLD-tree of rank n for $P^+; \leftarrow p$.*

Proof. Repeat the constructions from Lemmas 26, 27 and check that the resulting derivation or tree, is systematic and of the required rank. □

Lemma 36 *Let P be a propositional general program and let p be a propositional symbol. Then:*

1. *$SLD(P^+, p) = YES$ implies $SLDNF(P, p) = YES$,*
2. *$SLD(P^+, p) = NO$ implies $SLDNF(P, p) = NO$.*

Proof. By Lemma 34, it is enough to prove the following two statements:

1. If there is a successful systematic SLD-derivation of rank n for $P^+; \leftarrow p$, then there is a successful SLDNF-derivation for $P; \leftarrow p$.

2. If there is a systematic finitely failed SLD-tree of rank n for $P^+; \leftarrow p$, then there is a finitely failed SLDNF-tree for $P; \leftarrow p$.

We prove both statements in parallel, by induction on n.

Successful SLD-derivation for $P^+; \leftarrow p$ of rank 0.

No "negative" subgoals (like $\overline{q}$) were selected in this derivation. Therefore if we replace every $\overline{q}$ by $\neg q$, we will obtain a successful SLDNF-derivation for $P; \leftarrow p$.

Finitely failed SLD-tree for $P^+; \leftarrow p$ of rank 0.

No "negative" subgoals (like $\overline{q}$) were selected in the derivations of the tree. Therefore if we replace every $\overline{q}$ by $\neg q$, we will obtain a finitely failed SLDNF-tree for $P; \leftarrow p$.

Successful systematic SLD-derivation for $P^+; \leftarrow p$ of rank $n + 1$.

Consider any maximal lemma in this derivation which is negative. Rank of the lemma is $\leq n$. If the lemma starts with $\leftarrow \ldots \wedge \overline{q} \wedge \ldots$, where $\overline{q}$ is the selected atom, by Lemma 35 there is a systematic finitely failed SLD-tree for $P; \leftarrow q$ of rank $\leq n$, and by induction hypothesis there is a finitely failed SLDNF-tree for $P; \leftarrow q$. Therefore by cutting out all negative maximal lemmas from the SLD-derivation for $P^+; \leftarrow p$, and by replacing all the remaining "negative" atoms $\overline{q}$ by $\neg q$, we obtain a successful SLDNF-derivation for $P; \leftarrow p$.

Finitely failed systematic SLD-tree for $P^+; \leftarrow p$ of rank $n + 1$.

In this case we first have to deal with lemmas and than with failed lemmas.

Consider a node in the tree, at which a branch with a negative maximal lemma starts: $\leftarrow \ldots \wedge \overline{q} \wedge \ldots$ with the selected atom $\overline{q}$. The lemma is of rank $\leq n$, so by Lemma 35, there is a finitely failed systematic SLD-tree for $P^+; \leftarrow q$ of rank $\leq n$, and by induction hypothesis there is a finitely failed SLDNF-tree for $P; \leftarrow q$. Delete all the branches coming out of $\leftarrow \ldots \wedge \overline{q} \wedge \ldots$ except the one containing the lemma; in this branch cut out the lemma. Do that for all negative maximal lemmas.

Now consider a node where a negative maximal failed lemma starts: $\leftarrow \ldots \wedge \overline{q'} \wedge \ldots$. Notice that as we scooped all lemmas in the previous step, every branch coming out of our node is now a failed lemma. The tree from this node down, is of rank $\leq n$. By Lemma 35 there exists a successful systematic SLD-derivation for $P; \leftarrow q'$ of rank $\leq n$, and by induction hypothesis there exists a successful SLDNF-derivation for $P; \leftarrow q'$. By Proposition 11 there exists a finitely failed SLDNF-tree for $P; \leftarrow \neg q$; replace the tree starting at $\leftarrow \ldots \wedge \overline{q'} \wedge \ldots$ by this tree.

Finally replace all the negative atoms $\overline{q}$ in the tree by $\neg q$. Resulting tree is a finitely failed SLDNF-tree for $P; \leftarrow p$. □

Proof of Theorem 19. Points 1 and 2 result immediately from Lemmas 36 and 29. Point 3 follows from 1 and 2 by Proposition 10. (We apply Proposition 10 to both P and P^+.) □

Now we are ready to prove the theorem on soundness and completeness.

Proof of Theorem 20.

1. Notice that for any propositional symbol p,

   $SLDNF(P,p) = YES$, iff $T_{P+} \uparrow \omega \models p$, and
   $SLDNF(P,\neg p) = YES$, iff $T_{P+} \uparrow \omega \models \overline{p}$.

   Indeed:

   $SLDNF(P,p) = YES$, iff (by Theorem 19)
   $SLD(P^{+},p) = YES$, iff
   $T_{P+} \uparrow \omega \models p$

   and:

   $SLDNF(P,\neg p) = YES$, iff (by Proposition 11)
   $SLDNF(P,p) = NO$, iff (by Theorem 19)
   $SLD(P^{+},p) = NO$, iff (by Lemma 28)
   $SLD(P^{+},\overline{p}) = YES$, iff
   $T_{P+} \uparrow \omega \models \overline{p}$.

   Now consider arbitrary formula B. Assume that disjunctive normal form of B is $\bigvee_i \bigwedge_j L_{i,j}$. We have:

   $SLDNF(P,B) = YES$, iff
   $SLDNF(P,\bigvee_i \bigwedge_j L_{i,j}) = YES$, iff
   there is i such that $SLDNF(P,\bigwedge_j L_{i,j}) = YES$, iff
   there is i such that for all j, $SLDNF(P,L_{i,j}) = YES$, iff
   there is i such that for all j, $T_{P+} \uparrow \omega \models \langle L_{i,j}\rangle$, iff
   $T_{P+} \uparrow \omega \models \bigvee_i \bigwedge_j \langle L_{i,j}\rangle$, iff
   $Comp(P^{+}) \vdash \bigvee_i \bigwedge_j \langle L_{i,j}\rangle$, iff
   $Comp(P^{+}) \vdash B^{+}$.

2. We have:

   $SLDNF(P,B) = NO$, iff (by Proposition 11)
   $SLDNF(P,\neg B) = YES$, iff (by 1)
   $Comp(P^{+}) \vdash (\neg B)^{+}$.

3. From 1 and 2 by Proposition 10. □

3.3. Characterization of Propositional SLDNF-resolution in Intuitionistic and Intermediate Logics.

Our next task will be to generalize Theorem 20 on soundness and completeness, allowing deductions in, so called, *intermediate logics*. These are logics which fall between intuitionistic and classical logic (including the two).

Let us mention that in other research, Miller and Gabbay showed that intuitionistic logic rather than classical logic provides the correct semantics for logic programs.[14,21]

The propositional intuitionistic logic will be denoted **IL**. Every intuitionistic tautology is a classical tautology, but not vice versa; for instance $p \vee \neg p$ is not an intuitionistic tautology. Intuitionistic logic is considered to be a logic of constructive reasoning. Constructivists argue that in order to prove $B_1 \vee B_2$ one needs to prove at least one of $B_1,\ B_2$; in consequence there are propositions p for which $p \vee \neg p$ cannot be proved constructively. Much of common sense reasoning has a constructive flavor, and when formalized can be placed between intuitionistic and classical logic.

While considering declarative semantics of programs in intuitionistic logic we are allowed to replace P by P^C and vice versa. This is because the following formula is an intuitionistic tautology:

$$(B_0 \leftarrow B_1) \wedge (B_0 \leftarrow B_2) \equiv (B_0 \leftarrow B_1 \vee B_2).$$

However one needs to realize that many other transformations of formulas, commonly used in classical logic, are not valid in intuitionistic logic. Consider for example classical tautologies used to transform a formula to disjunctive normal form. Among them, the following are valid in intuitionistic logic:

$$\begin{array}{l} \neg(B_1 \wedge B_2) \leftarrow \neg B_1 \vee \neg B_2, \\ \neg(B_1 \vee B_2) \equiv \neg B_1 \wedge \neg B_2, \\ B_1 \vee (B_2 \wedge B_3) \rightarrow (B_1 \vee B_2) \wedge (B_1 \vee B_3), \\ B_1 \wedge (B_2 \vee B_3) \equiv (B_1 \wedge B_2) \vee (B_1 \wedge B_3), \end{array}$$

however the following are not intuitionistic tautologies:

$$\begin{array}{l} \neg(B_1 \wedge B_2) \rightarrow \neg B_1 \vee \neg B_2 \\ B_1 \vee (B_2 \wedge B_3) \leftarrow (B_1 \vee B_2) \wedge (B_1 \vee B_3). \end{array}$$

The following proposition is of crucial importance for this section.

Proposition 37 *Let P be a propositional positive program and let B be a propositional positive formula. Let $\mathbf{L}$ be any intermediate logic (including intuitionistic logic and classical logic). Then:*

1. *$P \vdash_{\mathbf{L}} B$ iff $P \vdash B$,*
2. *$Comp(P) \vdash_{\mathbf{L}} B$ iff $Comp(P) \vdash B$,*

where $\vdash$ is the consequence operation of classical logic.

Proof. The implications to the right are obvious. It is enough to prove implications to the left for $\mathbf{L}$ being intuitionistic logic.

Consider any Kripke interpretation M for *Prop*. (See any standard text for the definition.[9]) The nodes of M are subsets of *Prop*. The relation $w \Vdash B$ is defined for nodes w and formulas B. Every node w can be also treated as a classical interpretation of *Prop*, namely: $w(p) = T$ if $p \in w$, otherwise $w(p) = F$. We have:

$$w \Vdash B \text{ iff } w \models B, \text{ for any positive formula } B, \qquad \text{(i)}$$

From (i) we obtain:

$$w \Vdash B_1 \leftarrow B_2 \text{ implies } w \models B_1 \leftarrow B_2, \text{ for any positive formulas } B_1, B_2. \qquad \text{(ii)}$$

$$w \Vdash B_1 \equiv B_2 \text{ implies } w \models B_1 \equiv B_2, \text{ for any positive formulas } B_1, B_2. \qquad \text{(iii)}$$

Now, to prove point 1 of the proposition assume that $P \vdash B$. Take any Kripke model M for P. For any node w in M we have $w \Vdash P$; thus by (ii), $w \models P$; thus $w \models B$, thus by (i), $w \Vdash B$. Thus, $M \Vdash B$. Therefore $P \models_{\mathbf{IL}} B$. This proves the implication to the left in point 1 of the proposition.

The proof of the point 2 is analogous and uses implication (iii). □

Because of problems with disjunctive normal form which were mentioned earlier, we need to carefully consider whether we are allowed to turn programs and queries into such a form while working in intuitionistic logic. As the reader already expects, in general, P and P^{dnf} are not intuitionistically equivalent, even for positive programs. However if we are concerned with positive intuitionistic consequences of P and P^{dnf} it turns out that there is no difference between them! This is made precise in the following corollary which results from Proposition 37 and the fact that P, P^{dnf} (and other transformations of P) are equivalent in classical logic.

Corollary 38 *Let P be a propositional positive program and let B be a propositional positive formula. Let $\mathbf{L}$ be any intermediate logic (including intuitionistic logic and classical logic). Then:*

1. *Every two of the following are equivalent: $P \vdash_{\mathbf{L}} B$, $P^C \vdash_{\mathbf{L}} B$, $P^{deM} \vdash_{\mathbf{L}} B^{deM}$, $P^{dnf} \vdash_{\mathbf{L}} B^{dnf}$, $P^S \vdash_{\mathbf{L}} B^{dnf}$,*
2. *Every two of the following are equivalent: $Comp(P) \vdash_{\mathbf{L}} B$, $Comp(P^C) \vdash_{\mathbf{L}} B$, $Comp(P^{deM}) \vdash_{\mathbf{L}} B^{deM}$, $Comp(P^{dnf}) \vdash_{\mathbf{L}} B^{dnf}$, $Comp(P^S) \vdash_{\mathbf{L}} B^{dnf}$.*

The following corollary results from Proposition 37 and the fact that in classical logic, for positive programs and goals, $Comp(P) \vdash B$ iff $P \vdash B$.

Corollary 39 *Let P be a propositional positive program and let B be a propositional positive formula. Let $\mathbf{L}$ be any intermediate logic (including intuitionistic logic and classical logic). Then $Comp(P) \vdash_{\mathbf{L}} B$ iff $P \vdash_{\mathbf{L}} B$.*

As in programming practice, reasoning about SLDNF-derivations for bigger programs is not expected to be formal, it is important to know that the entire machinery of classical logic is not necessary in the proofs. So the following result assures us that there is a good chance to deduce all the needed properties of propositional SLDNF-derivations while reasoning in an informal way.

Theorem 40 (Soundness and completeness)
Let $\mathbf{L}$ be any intermediate logic (including intuitionistic logic and classical logic). Let $P; \leftarrow B$ be a propositional general program and goal. Then:

1. *$SLDNF(P, B) = YES$ iff $Comp(P^+) \vdash_{\mathbf{L}} B^+$.*
2. *$SLDNF(P, B) = NO$ iff $Comp(P^+) \vdash_{\mathbf{L}} (\neg B)^+$.*
3. *$SLDNF(P, B) = \uparrow$ iff $Comp(P^+) \not\vdash_{\mathbf{L}} B^+$ and $Comp(P^+) \not\vdash_{\mathbf{L}} (\neg B)^+$.*

Notice that by Corollary 39 we could use P^+ instead of $Comp(P^+)$ in this theorem.

Proof. Immediately from Theorem 20 and Proposition 37. □

3.4. Modal Completion and Characterization of Propositional SLDNF-resolution in Modal Logics

In this section we will define a notion of modal completion of a general program. We will obtain a soundness and completeness for propositional SLDNF-resolution with respect to a wide family of modal logics. Among them there will be some well known logics of knowledge including a version of the system **S4**.

Let us mention that in another research, Gabbay proved completeness of propositional SLDNF-resolution using a notion of a fix-point completion in the propositional modal logic of arithmetical provability.[13] A similar characterization for stratified first-order programs was obtained by Balbiani.[2]. The modal logic of arithmetical provability is rather complex and not widely known, our approach will be more straightforward.

The intuitions behind the construction of modal completion are the following. Consider an agent who performs SLDNF-resolution with a fixed program P and various goals. While considering a positive goal the agent uses breadth-first search in the tree of SLDNF-derivations, so if a successful SLDNF-derivation exists, sooner or later he will find it. While considering a negative goal, the agent builds in parallel all possible failure trees, so if a finite failure tree exists, sooner or later he will find it. The agent is a perfect executor of SLDNF-resolution: whenever the goal is SLDNF-derivable, the agent will derive it.

We will characterize the SLDNF-derivability, as performed by the agent, in modal logic. $\Box p$ will mean *the agent can prove p.*

Let $\sim$ stand for negation as failure. To prove $\sim p$ the agent constructs a finite failure tree for p. Every branch of this tree can be viewed as a failed attempt to construct a successful derivation for p. The branches of the tree exhaust all possibilities: if all branches are failed, the agent can be sure that there is no successful derivation for p (even if a different order of selecting literals were used). So, by constructing a finite failure tree for p the agent *proves* that using his limited proof methods he will *never* be able to *prove* p. Thus $\sim p$ will be interpreted as $\Box\neg\Box p$.

Our characterization will of course depend on the program P considered by the agent. If there is a fact p (i.e. statement $p \leftarrow \top$) in the program, our axioms, denoted $P^{\Box}$, will imply $\Box p$. If there is a statement $p_1 \leftarrow p_2 \vee (\neg p_3 \wedge p_4)$ in P^C, the axioms $P^{\Box}$ will imply $\Box p_1 \leftarrow \Box p_2 \vee (\Box\neg\Box p_3 \wedge \Box p_4)$ as well as
$\Box\neg\Box p_1 \leftarrow \Box\neg\Box p_2 \wedge (\Box p_3 \vee \Box\neg\Box p_4)$. These formulas accurately describe what the agent can prove. We can expect the following equivalences

$$SLDNF(P,p) = YES \text{ iff } P^{\Box} \vdash_{\mathbf{L}} \Box p,$$
$$SLDNF(P,p) = NO \text{ iff } P^{\Box} \vdash_{\mathbf{L}} \Box\neg\Box p.$$

We will also consider a set of axioms $C(P^{\Box})$ obtained from $P^{\Box}$ by replacing reversed implications $\leftarrow$ by equivalences $\equiv$. Unlike $P^{\Box}$ the axioms $C(P^{\Box})$ imply not only what the agent can prove but also what he cannot prove. The set $C(P^{\Box})$ is expected to give an analogous characterization of SLDNF-resolution.

It would be interesting to see what modal logics $\mathbf{L}$ can be used in such characterizations. A choice of a modal logic $\mathbf{L}$ will affect how much reasoning power the agent has besides being able to perform SLDNF-resolution. For instance, $\mathbf{L}$ may contain the axiom schema $\Box B \to \Box\Box B$ which is known in the domain of logics of knowledge as the axiom of positive introspection. However, we will see that $\mathbf{L}$ cannot contain the the axiom of negative introspection: $\neg\Box B \to \Box\neg\Box B$.

And now, a formal treatment of this idea.

Definition 41 (Modal transformation and completion)
The *modal transformation* $(\cdot)^{\Box}$ of propositional formulas and general programs is defined as follows. For a formula B over *Prop* containing no connectives other than $\top, \bot, \wedge, \vee, \neg$, the formula $B^{\Box}$ is obtained from B^{+} by replacing every propositional symbol p by $\Box p$ and every symbol $\bar{p}$ by $\Box\neg\Box p$. For a general program P, its modal

transformation $P^{\Box}$ is obtained from P^+ by replacing every propositional symbol p by $\Box p$ and every symbol $\bar{p}$ by $\Box\neg\Box p$.

The *modal completion* $C(P^{\Box})$ is obtained from $P^{\Box}$ by replacing every reversed implication $\leftarrow$ by an equivalence $\equiv$. $\Box$

Example 42 Consider the following general program P:

$$p \leftarrow p$$
$$q \leftarrow p$$
$$q \leftarrow \neg q$$

We will construct its modal transformation $P^{\Box}$ and modal completion $C(P^{\Box})$.
P^+ is:

$$p \leftarrow p \qquad\qquad \bar{p} \leftarrow \bar{p}$$
$$q \leftarrow p \vee \bar{q} \qquad\qquad \bar{q} \leftarrow \bar{p} \wedge q$$

Thus $P^{\Box}$ is:

$$\Box p \leftarrow \Box p \qquad\qquad \Box\neg\Box p \leftarrow \Box\neg\Box p$$
$$\Box q \leftarrow \Box p \vee \Box\neg\Box q \qquad\qquad \Box\neg\Box q \leftarrow \Box\neg\Box p \wedge \Box q$$

And $C(P^{\Box})$ is:

$$\Box p \equiv \Box p \qquad\qquad \Box\neg\Box p \equiv \Box\neg\Box p$$
$$\Box q \equiv \Box p \vee \Box\neg\Box q \qquad\qquad \Box\neg\Box q \equiv \Box\neg\Box p \wedge \Box q$$

We define *modal intuitionistic logic* **MIL** as the logic with whose axioms are all modal instances of propositional intuitionistic tautologies and the only inference rule is modus ponens.

We define **dS4** to stand for a version of the system **S4** in which the use of the rule of necessitation is allowed only in proofs of logical theorems. More exactly, the logic **dS4** is axiomatized using the following logical axiom schemas and rules of inference:

CL	logical axiom schemas of the classical propositional calculus,
K	$\Box(B_1 \rightarrow B_2) \rightarrow (\Box B_1 \rightarrow \Box B_2)$,
T	$\Box B \rightarrow B$,
4	$\Box B \rightarrow \Box\Box B$,
MP	$B_1, B_1 \rightarrow B_2 \ / \ B_2$,
RN	$B \ / \ \Box B$.

Given a set of formulas $\Gamma \cup \{B\}$ we write $\Gamma \vdash_{\mathbf{dS4}} B$ if there exists a sequence of formulas satisfying the following conditions. The sequence consists of a global part followed by a local part; either part can be empty. The sequence ends with B. In the *global part* every formula is either an instance of a logical axiom schema or results from formulas occurring earlier in the sequence by means of rules MP or RN. In the *local part* every formula is either a member of Γ, or results from formulas occurring earlier in the sequence (possibly in the global part) by means of the rule MP.

Notice that in this logic the use of RN is restricted, causing that for instance, $p \not\vdash_{\mathbf{dS4}} \Box p$.

The logic **dS4** is sound and complete with respect to the class of Kripke interpretations with origins, based on reflexive and transitive frames. Recall that a *Kripke interpretations with an origin* M is a Kripke interpretation in which one world w_0 has been distinguished. A formula B is *valid* in M if $w_0 \Vdash B$; this is denoted by $M \models B$.

Let us comment on the informal interpretation of the axiomatization of this logic. The rule RN is called the rule of necessitation. This rule allows us to derive $\Box B$ for any logical theorem B. This means that the agent can prove every logical theorem. In the context of CL and MP, schema K can be rewritten as

$$(\Box B_1 \wedge \Box(B_1 \rightarrow B_2)) \rightarrow \Box B_2$$

and interpreted as: if the agent can prove B_1 and can prove $B_1 \rightarrow B_2$ then he can prove B_1. This means that the knowledge of the agent is closed under the rule of modus ponens MP. Schema T can be read as: if the agent can prove B, then B is true, so the agent's proof methods are sound. Thus, schemas CL, MP, RN, K, T imply that the agent is a perfect and sound reasoner. Schema 4 can be read as: if the agent can prove B is then he can prove that he can prove B. So the agent can reason not only about notions external to him, but also about his own knowledge. The property expressed by schema 4 is called a positive introspection.

Theorem 43 (Soundness and completeness)
Let **L** *be any modal logic extending* **MIL** *and contained in* **dS4**. *Let* $P; \leftarrow B$ *be a propositional general program and goal. Then:*

1. *SLDNF*$(P, B) =$ *YES* *iff* $C(P^{\Box}) \vdash_{\mathbf{L}} B^{\Box}$,
2. *SLDNF*$(P, B) =$ *NO* *iff* $C(P^{\Box}) \vdash_{\mathbf{L}} (\neg B)^{\Box}$,
3. *SLDNF*$(P, B) = \uparrow$ *iff* $C(P^{\Box}) \not\vdash_{\mathbf{L}} B^{\Box}$ and $C(P^{\Box}) \not\vdash_{\mathbf{L}} (\neg B)^{\Box}$.

Example 44 Continuing Example 42.
Notice that *SLDNF*$(P, p) =\uparrow$. As *Comp*$(P) \vdash p$ this program constitutes another example showing that SLDNF-resolution is not complete with respect to Clark's completion of P. Let us now see how Theorem 43 works for this program. According to the theorem in order to show that *SLDNF*$(P, p) =\uparrow$ it is enough to show that in **dS4** $C(P^{\Box}) \not\vdash \Box p$ and $C(P^{\Box}) \not\vdash \Box\neg\Box p$. For that, we present a Kripke model for **dS4**. Let $M = \langle W, w_0, R, v\rangle$ where $W = \{w_0, w_1\}$, R is a relation containing pairs $\langle w_0, w_0\rangle$, $\langle w_0, w_1\rangle$, $\langle w_1, w_1\rangle$, and

$$v(w_0, p) = F, \qquad v(w_0, q) = F,$$
$$v(w_1, p) = T, \qquad v(w_1, q) = T,$$

In this model $C(P^{\Box})$ is valid and neither $\Box p$ nor $\Box\neg\Box p$ are valid. $\Box$

Now let us state a result which implies that in the equivalences of Theorem 43 we could use just $P^{\Box}$ instead of $C(P^{\Box})$.

Theorem 45 *Let* **L** *be any modal logic extending* **MIL** *and contained in* **dS4**. *Let* $P; \leftarrow B$ *be a propositional general program and goal. Then:*

$$C(P^{\Box}) \vdash_{\mathbf{L}} B^{\Box} \quad \textit{iff} \quad P^{\Box} \vdash_{\mathbf{L}} B^{\Box}.$$

Remark 46 The reasoning which is used in the proof does not exclude the possibility that a logic **L** which is not contained in **dS4** could be used in of Theorem 43. Let us notice however that if such a logic **L** contains CL and MP it cannot contain the schema 5: $\neg\Box B \rightarrow \Box\neg\Box B$. If schema 5 belonged to the logic the equivalences of Theorem 43 would not hold. This can be shown using program P containing statements $p \leftarrow p$ and $q \leftarrow p \vee \neg p$. We have *SLDNF*$(P, q) =\uparrow$ On the other hand $P^{\Box}$ contains formula $\Box q \leftarrow \Box p \vee \Box\neg\Box p$, from which, using CL, MP and schema 5 one can derive $\Box q$.

The property expressed by schema 5 is called negative introspection. Let us think again about the informal interpretation of $\Box$ discussed at the beginning of this section. Allowing the axiom of negative introspection in **L** would mean that if there is no successful derivation for p the agent somehow deduces $\sim p$ even if there is no *finite* failure tree. This is clearly contrary to the spirit of SLDNF. $\Box$

Remark 47 Let us use the symbol **sS4** to denote the version of logic **S4** which is a standard system in the sense of Rasiowa. In **sS4** items CL, K, T, 4 are used as logical axiom schemas and the uses of rules MP and RN are unrestricted. This means, $\Gamma \vdash_{\mathbf{sS4}} B$ if there exists a sequence of formulas ending with B in which every formula is either an instance of a logical axiom schema, or a member of Γ, or results from formulas occurring earlier in the sequence by means of rules MP or RN.

Unlike for **dS4**, we have $p \vdash_{\mathbf{sS4}} \Box p$.

The logic **sS4** is sound and complete with respect to the class of Kripke interpretations based on reflexive and transitive frames in which validity is defined as: $M \models B$ if $w \Vdash B$ for every world w of M.

Notice that logic **sS4** cannot be used in the equivalences of Theorem 43. This can be shown by considering program P containing one statement: $p \leftarrow \neg p$. We have $\mathit{SLDNF}(P,p) = \uparrow$. On the other hand, $P^{\Box}$ implies the formula $\Box p \equiv \Box\neg\Box p$ which does not have any **sS4** model. So $P^{\Box} \models_{\mathbf{sS4}} \Box p$.

The fact that **dS4** but not **sS4** can be used in Theorem 43 adds the following to our informal understanding of the modal interpretation. What the agent can or cannot prove is accurately described by $C(P^{\Box})$, however the agent cannot know (i.e. prove) that his knowledge is described by $C(P^{\Box})$. $\Box$

3.5. Proofs

To prove Theorems 43 and 45 it is enough to show the following implications:

(i)	$\mathit{SLDNF}(P,B) = \mathit{YES}$	implies	$P^+ \vdash B^+$
(ii)	$P^+ \vdash B^+$	implies	$P^{\Box} \vdash_{\mathbf{L}} B^{\Box}$
(iii)	$P^{\Box} \vdash_{\mathbf{L}} B^{\Box}$	implies	$C(P^{\Box}) \vdash_{\mathbf{L}} B^{\Box}$
(iv)	$C(P^{\Box}) \vdash_{\mathbf{L}} B^{\Box}$	implies	$\mathit{Comp}(P^+) \vdash B^+$
(v)	$\mathit{Comp}(P^+) \vdash B^+$	implies	$\mathit{SLDNF}(P,B) = \mathit{YES}$

These implications suffice for the proof of Theorem 45 and point 1 of Theorem 43. Point 2 of Theorem 43 can be proved analogously; point 3 results from points 1,2.

Implications (i) and (v) result from Theorem 20. Implication (ii) is first proved for B being a propositional symbol p by induction on n such that $p \in T_P \uparrow n$. Then the statement is generalized to arbitrary formula B. This is the part of the proof where it is important that $\mathbf{MIL} \subseteq \mathbf{L}$. Implication (iii) is obvious. For the proof of implication (iv) we assume $\mathit{Comp}(P^+) \not\vdash B^+$ and construct a Kripke model for **dS4** which is a model for $C(P^{\Box})$ but not for $B^{\Box}$. This is the part of the proof where it is important that $\mathbf{L} \subseteq \mathbf{dS4}$.

Now, the details.

Definition 48 Let P be a set of definite clauses. If $(q \leftarrow \top) \in P$ then the sequence consisting of this single clause is called a *pure detachment proof of rank 0 of q from P*. If $(q \leftarrow p_1 \wedge \ldots \wedge p_m) \in P$ and $\mathcal{P}_1, \ldots, \mathcal{P}_m$ are pure detachment proofs of ranks $\leq n$

of, respectively, $p_1, \ldots, p_m$ from P, then the concatenation of sequences $\mathcal{P}_1, \ldots, \mathcal{P}_m$ and a one-element sequence $\langle q \leftarrow p_1 \wedge \ldots \wedge p_m \rangle$ is called a *pure detachment proof of rank* $n+1$ *of* q *from* P. $\Box$

Notice that by inserting some tautologies, any pure detachment proof can be turned into a formal proof in the propositional classical logic.

Lemma 49 *Let* P *be a positive program, and let* p *be a propositional symbol. If* $P \vdash p$ *then* p *has a pure detachment proof from* P.

Proof. If $P \vdash p$, then $T_P \uparrow \omega \vdash p$. Thus for a certain $n < \omega$, we have $T_P \uparrow n \vdash p$. By induction on n one can show that for any propositional symbol q, $T_P \uparrow n \vdash q$ iff q has a pure detachment proof of rank n from P. $\Box$

Lemma 50 *Let* $\mathbf{L}$ *be a propositional modal logic extending* **MIL**. *Let* $P; \leftarrow B$ *be a propositional general program and goal. Then:*

1. $P^+ \vdash B^+$ *implies* $P^\Box \vdash_{\mathbf{L}} B^\Box$.
2. $P^+ \vdash (\neg B)^+$ *implies* $P^\Box \vdash_{\mathbf{L}} (\neg B)^\Box$.

Proof. If B is a literal, the implication in point 1 results from Lemma 49.
If B is a formula in disjunctive normal form $\bigvee_i \bigwedge_j L_{i,j}$, we have

$P^+ \vdash B^+$, iff
there is i such that for all j, $Comp(P^+) \vdash L_{i,j}$, implies
there is i such that for all j, $P^\Box \vdash_{\mathbf{L}} L^\Box_{i,j}$, iff
$P^\Box \vdash_{\mathbf{L}} B^\Box$.

If B is arbitrary formula, we have

$P^+ \vdash B^+$, iff
$P^+ \vdash (B^{dnf})^+$, implies
$P^\Box \vdash_{\mathbf{L}} (B^{dnf})^\Box$, iff
$P^\Box \vdash_{\mathbf{L}} (B^\Box)^{dnf}$, iff
$P^\Box \vdash_{\mathbf{L}} B^\Box$,

the last equivalence being justified by Theorem 38.
This proves point 1. Point 2 follows from point 1. $\Box$

Lemma 51 *Let* $\mathbf{L}$ *be a modal logic contained in* **dS4**. *Let* $P; \leftarrow B$ *be a propositional general program and goal. Then:*

1. $C(P^\Box) \vdash_{\mathbf{L}} B^\Box$ *implies* $Comp(P^+) \vdash B^+$,
2. $C(P^\Box) \vdash_{\mathbf{L}} (\neg B)^\Box$ *implies* $Comp(P^+) \vdash (\neg B)^+$.

Proof. We will prove that $Comp(P^+) \not\vdash B^+$ implies $C(P^\Box) \not\vdash_{\mathbf{dS4}} B^\Box$.
Assume $Comp(P^+) \not\vdash B^+$. Let v_0 be the least fix-point of T_{P^+}.
Consider a Kripke interpretation $M = \langle W, w_0, R, v \rangle$ for **dS4**. We set $W = \{w_0, w_1\}$, R to be a relation containing pairs $\langle w_0, w_0 \rangle$, $\langle w_0, w_1 \rangle$, $\langle w_1, w_1 \rangle$, and

if $v_0(p) = T$ and $v_0(\bar{p}) = F$ then we set $v(w_0, p) = T$ and $v(w_1, p) = T$,
if $v_0(p) = F$ and $v_0(\bar{p}) = T$ then we set $v(w_0, p) = F$ and $v(w_1, p) = F$,
if $v_0(p) = F$ and $v_0(\bar{p}) = F$ then we set $v(w_0, p) = F$ and $v(w_1, p) = T$.

(This definition is complete because for no p, $v_0(p) = T$ and $v_0(\bar{p}) = T$.)
Let B be any formula over *Prop* built from literals and truth constants using $\wedge, \vee$.
We claim:

1. $w_0 \Vdash B^\Box$ implies $w_1 \Vdash B^\Box$,

2. $w_0 \Vdash B^\Box$ iff $M \models B^\Box$,
3. $v_0 \models B^+$ iff $w_0 \Vdash B^\Box$,
4. $v_0 \models B^+$ iff $M \models B^\Box$.

Implication (1) can be shown by induction on B. Equivalence (2) results from (1). Equivalence (3) can be shown by induction on B. Equivalence (4) results from (2) and (3).

Equivalence (4) implies that for any any formula over *Prop* built using $\top, \bot, \wedge, \vee, \neg$, we have

$$v_0 \models B^+ \text{ iff } M \models B^\Box. \tag{$*$}$$

As $v_0 \models Comp(P^+)$, equivalence $(*)$ implies that $M \models C(P^\Box)$. As $v_0 \not\models B^+$, equivalence $(*)$ implies that $M \not\models B^\Box$. Thus $C(P^\Box) \not\models B^\Box$ and $C(P^\Box) \not\vdash_{\mathbf{dS4}} B^\Box$. This ends the proof of point 1 of the lemma; point 2 follows from point 1. □

Proof of Theorem 43. Points 1 and 2 follow from Theorem 20 and Lemmas 50, 51. Point 3 follows from 1 and 2 by Proposition 10. □

Proof of Theorem 45. From Lemmas 50, 51 and Corollary 39. □

4. Constructive Transformation versus Multi-valued Semantics

Fitting proposed using interpretations based on Kleene's strong three-valued logic as a semantics of logic programs.[10] Then he considered semantics based on the four-valued logic of Belnap and also on some more general structures.[11,5] Stark considered three-valued semantics and made a connection between the three-valued semantics, a variant of the four-valued semantics, and the two-valued semantics.[27,28] We will give relevant definitions and show other connections, making a complete comparison of these semantics.

All the results of this section can be generalized to the first-order case.

4.1. Two-valued (Classical) Semantics

Although the classical propositional logic and two-valued semantics of logic programs are well known, we will need to specify some of the definitions in order to fix the terminology and make notational conventions precise.

The *formulas* of classical propositional logic are built in the usual way from propositional symbols, *connectives:* $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$, and parentheses.

The *classical truth values:* true and false, called also *Boolean truth values* will be represented by 0 and 1 respectively. The operations on $\{0, 1\}$ corresponding to the connectives are defined as usual, and will be denoted by the same symbols as the connectives. By a *two valued interpretation* or a *classical interpretation* we understand any mapping v form the set of propositional symbols into the set $\{0, 1\}$. The operations $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$ allow us to extend the interpretation v from propositional symbols to arbitrary propositional formulas. Given a propositional formula B and a two-valued interpretation v we say that B *is true in* v or that v *is a model for* B if $v(B) = 1$; this will be denoted by $v \models B$, or by $v \models_2 B$ if we wish to emphasize that v is two-valued. Given a set of propositional formulas

$\Gamma \cup \{B\}$, we write $\Gamma \models B$ or $\Gamma \models_2 B$ to represent that every two-valued model for Γ is a model for B. We write $\models B$ or $\models_2 B$ to represent that B is true in every two-valued interpretation.

We write $v_1 \leq v_2$ if for every propositional symbol p, $v_1(p) \leq v_2(p)$. Notice that, for every formula B containing no connectives other than $\top, \bot, \wedge, \vee$,

if $v_1 \leq v_2$ and $v_1 \models_2 B$ then $v_2 \models_2 B$, and
if $v_1 \leq v_2$ and $v_2 \models_2 \neg B$ then $v_1 \models_2 \neg B$;

these properties will be referred to as the *monotonicity of B*. The family of all two-valued interpretations of a fixed propositional language with the ordering $\leq$ forms a complete lattice. With any positive propositional program P we associate an operator on this lattice:

$$T_P(v) = \{p \mid \text{ there exists } B \text{ s.t. } (p \leftarrow B) \in P \text{ and } v \models_2 B\}$$

(Notice that transforming P does not change the outcome of the operator: $T_P(v) = T_{P^C}(v) = T_{P^{deM}}(v) = T_{P^{dnf}}(v) = T_{P^S}(v)$.) For a program P in Clark's form, for any propositional symbol p we have:

$$(T_P(v))(p) = v(B)$$

where B is the body of the unique statement defining p in P.
It is known that

$$T_P(v) = v \text{ iff } v \models_2 Comp(P).$$

4.2. Four-valued Semantics

We will consider a four-valued logic with the semantics involving the following *truth values*:

$T = \langle 1, 0\rangle$ (true),
$F = \langle 0, 1\rangle$ (false),
$U = \langle 0, 0\rangle$ (undefined),
$C = \langle 1, 1\rangle$ (contradictory).

Intuitively, these truth values summarize possible opinions of independent experts: if all the experts agree that p is true, we assign T to p; if all the experts agree that p is false, we assign F to p; if the experts did not give any opinion about p, we assign U to p; if some experts claim that p is true while others claim that it is false, we assign C to p. This intuition underlies definitions of operations $\top, \bot, \wedge, \vee, \neg$ which will be introduced.

As indicated above, the truth values T, F, U, C are identified with ordered pairs of Boolean values. Intuitively, the first component of the pair is the degree of truth: 1 if we have any evidence of truth, 0 otherwise; the second component is the degree of falsity: 1 if we have any evidence of falsity, 0 otherwise.
We define a function $^+ : \{T, F, U, C\} \rightarrow \{0, 1\}$ as $\langle z, z'\rangle^+ = z$.
We define a function $^- : \{T, F, U, C\} \rightarrow \{0, 1\}$ as $\langle z, z'\rangle^+ = z'$.
This means that the first component of the pair representing truth value x is x^+, and the second one is x^-.

The *truth ordering* $\leq_t$ and *knowledge ordering* $\leq_k$ are defined as follows:

$\langle x^+, x^-\rangle \leq_t \langle y^+, y^-\rangle$ iff $x^+ \leq y^+$ and $x^- \geq y^-$,
$\langle x^+, x^-\rangle \leq_k \langle y^+, y^-\rangle$ iff $x^+ \leq y^+$ and $x^- \leq y^-$.

So, the truth values are ordered as in the following diagram.

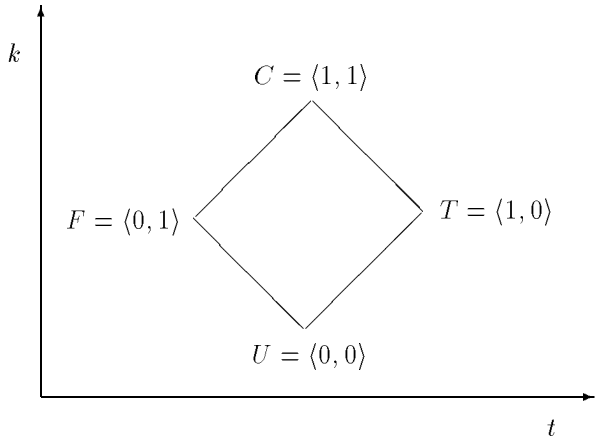

Fig. 1. Two orderings on the set of four truth-values.

Operations $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$ on $\{T, F, U, C\}$ are defined as follows:

$\top = \langle 1, 0 \rangle$,

$\bot = \langle 0, 1 \rangle$,

$\langle x^+, x^- \rangle \wedge \langle y^+, y^- \rangle = \langle x^+ \wedge y^+, x^- \vee y^- \rangle$,

$\langle x^+, x^- \rangle \vee \langle y^+, y^- \rangle = \langle x^+ \vee y^+, x^- \wedge y^- \rangle$,

$\neg \langle x^+, x^- \rangle = \langle x^-, x^+ \rangle$,

$\langle x^+, x^- \rangle \equiv \langle y^+, y^- \rangle = \langle (x^+ \equiv y^+) \wedge (x^- \equiv y^-), \neg((x^+ \equiv y^+) \wedge (x^- \equiv y^-)) \rangle$,

$\langle x^+, x^- \rangle \leftarrow \langle y^+, y^- \rangle = \langle (x^+ \leftarrow y^+) \wedge (x^- \leftarrow y^-), \neg((x^+ \leftarrow y^+) \wedge (x^- \leftarrow y^-)) \rangle$.

The operations can be also characterized by the following tables.

Table 1. Semantics of the connectives of the four-valued logic.

$\wedge$	T	F	U	C
T	T	F	U	C
F	F	F	F	F
U	U	F	U	F
C	C	F	F	C

$\vee$	T	F	U	C
T	T	T	T	T
F	T	F	U	C
U	T	U	U	T
C	T	C	T	C

	$\neg$
T	F
F	T
U	U
C	C

$\equiv$	T	F	U	C
T	T	F	F	F
F	F	T	F	F
U	F	F	T	F
C	F	F	F	T

$\leftarrow$	T	F	U	C
T	T	F	T	F
F	F	T	T	F
U	F	F	T	F
C	T	T	T	T

The operation $\leftarrow$ is not commutative; in the table specifying the values of $x \leftarrow y$, the values of x are listed in the border column and the values of y are listed in the border row. Notice that $\leftarrow$ restricted to the set $\{T, F\}$ does not coincide with the classical reversed implication, it coincides with the classical equivalence. This operation is different from the implication used by most of other authors.[1]

Many other operations could be defined, but these are the only ones we will need in our considerations.

Notice that:

$$
\begin{aligned}
&x \wedge y = glb_t\{x, y\},\\
&x \vee y = lub_t\{x, y\},\\
&\text{if } y \leq_k x \text{ then } x \leftarrow y = T, \text{ otherwise } x \leftarrow y = F,\\
&\text{if } x = y \text{ then } x \equiv y = T, \text{ otherwise } x \equiv y = F.
\end{aligned}
$$

The *formulas* of the four valued logic are built in the usual way from propositional symbols, *connectives:* $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$ and parentheses. We will use the same symbols for connectives and corresponding operations on $\{T, F, U, C\}$. A *four-valued interpretation* is any function I from the set of propositional symbols into the set $\{T, F, U, C\}$. For a four-valued interpretation I we define:

$$
\begin{aligned}
I^+ &= \{p \mid I(p)^+ = 1\} = \{p \mid I(p) = T \text{ or } I(p) = C\},\\
I^- &= \{p \mid I(p)^- = 1\} = \{p \mid I(p) = F \text{ or } I(p) = C\}.
\end{aligned}
$$

Notice that the pair $\langle I^+, I^-\rangle$ is an equivalent representation of I; the function I can be reconstructed from the pair $\langle I^+, I^-\rangle$ as $I(p) = \langle \chi_{I^+}(p), \chi_{I^-}(p)\rangle$ (where χ_S is the characteristic function of the set S: $\chi_S(x) = 1$ if $x \in S$, otherwise $\chi_S(x) = 0$).

The interpretation I extends to propositional formulas with connectives $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$ using the corresponding operations on $\{T, F, U, C\}$ defined above. Given a propositional formula B and a four-valued interpretation I we say that B *is true in* I, or that I *is a model for* B if $I(B) = T$; this will be denoted by $I \models B$, or by $I \models_4 B$ if we wish to emphasize that I is four-valued. Given a set of propositional formulas $\Gamma \cup \{B\}$, we write $\Gamma \models_4 B$ to represent that every four-valued model for Γ is a model for B. We write $\models_4 B$ to represent that B is true in every four-valued interpretation.

To get some feeling for this logic, note that if B_1 and B_2 are formulas containing no connectives other than $\top, \bot, \wedge, \vee$ then $\models_4 B_1 \equiv B_2$ iff $\models_2 B_1 \equiv B_2$. So, the associativity, distributivity and absorption laws hold in this four-valued logic. One can verify that also the de Morgan laws hold, but the law of excluded middle does not. Notice also that in every four-valued interpretation I:

$$
\begin{aligned}
&I(B_1 \equiv B_2) = I((B_1 \leftarrow B_2) \wedge (B_2 \leftarrow B_1)),\\
&I(\neg B_1 \leftarrow \neg B_2) = I(B_2 \leftarrow B_1),\\
&I((B \leftarrow B_1) \wedge (B \leftarrow B_2)) = I(B \leftarrow (B_1 \vee B_2)).
\end{aligned}
$$

The last of these equalities assures that Clark's transformation gives a program equivalent in this four-valued logic.

We define a partial order between four-valued interpretations, according to the increasing knowledge of what is true and what is false:

$$I_1 \leq_k I_2 \text{ iff } I_1^+ \subseteq I_2^+ \text{ and } I_1^- \subseteq I_2^-.$$

Notice that, for every formula B containing no connectives other than $\top, \bot, \wedge, \vee, \neg$

$$\text{if } I_1 \leq_k I_2 \text{ and } I_1 \models B \text{ then } I_2 \models B;$$

this property will be refereed to as the *k-monotonicity of* B.

The family of all four-valued interpretations of a fixed propositional language with the knowledge ordering $\leq_k$ forms a complete lattice. With any propositional general program P we associate an operator on this lattice: $\Phi_P(I) = \langle \Phi_P^+(I), \Phi_P^-(I)\rangle$ where:

$$\Phi_P^+(I) = \{p \mid \text{ there exists } B \text{ s.t. } (p \leftarrow B) \in P \text{ and } I(B)^+ = 1\},$$
$$\Phi_P^-(I) = \{p \mid \text{ for all } B, \text{if } (p \leftarrow B) \in P \text{ then } I(B)^- = 1\};$$

(Notice that transforming P does not change the outcome of the operator: $\Phi_P(v) = \Phi_{P^C}(v) = \Phi_{P^{deM}}(v) = \Phi_{P^{dnf}}(v) = \Phi_{P^S}(v)$.) For a general program P in Clark's form, for any propositional symbol p we have:

$$(\Phi_P(I))(p) = I(B)$$

where B is the body of the unique statement defining p in P.
It can be shown that:

$$\Phi_P(I) = I \text{ iff } I \models \mathit{Comp}(P),$$
$$\Phi_P(I) \leq_k I \text{ iff } I \models P.$$

The credit for various definitions presented in this subsection goes to other authors. The idea of truth values which summarize opinions of independent experts and the definitions of operations $\wedge, \vee$ are due to Belnap, however his notion of negation is different from $\neg$, and he did not consider $\leftarrow, \equiv$.[5] The definitions of $\equiv$ and Φ_P are obvious extensions of those introduced by Fitting in the context of three-valued logic.[10] The operation $\neg$ in the context of four-valued logic was explicitly defined by Fitting.[11] The operation $\leftarrow$ is an obvious extension of that introduced by Stärk in the context of three-valued logic.[27] Our semantical consequence relation $I \models_4 B$ is different from that of Stärk who defined his by the condition $I(B)^+ = 1$ (which is equivalent to: $I(B) = T$ or $I(B) = C$).[28]

4.3. Three-valued Semantics

The four-valued setting sketched above can be viewed as auxiliary. The area of "multi-valued semantics" for logic programming is predominantly concerned with the setting obtained by restricting the truth values to $\{T, F, U\}$. Here are the details.

Notice that the set $\{T, F, U\}$ is closed under the operations $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$ defined above. We will consider a three-valued logic in which the *truth values* are: T, F, U, and the relations $\leq_t, \leq_k$ and operations $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$ are restrictions of those from the semantics of the four valued logic.

So restricted operations $\top, \bot, \wedge, \vee, \neg$ are identical to those in Kleene's strong three-valued logic, and only the operations $\equiv, \leftarrow$ are different. Let us mention that Kleene's intention was to construct a logic whose semantics besides classical truth values (now denoted by T and F), contains an additional value U, which means that the classical truth value is not yet known. U should be thought of as a value which may turn into T or F when our knowledge about the world increases. This understanding of U implies that e.g. $U \wedge T = U$ and $U \vee T = T$. We set $U \wedge T = U$ because, in classical logic, the (classical) truth value of $p \wedge T$ cannot be determined if the (classical) truth value of p is not known. We set $U \vee T = T$ because, in classical logic, the truth value of $p \vee T$ is T independently of the (classical) truth value of p. All the cases in the definitions of $\top, \bot, \wedge, \vee, \neg$ in the logic we consider can be justified in this way.

By a *three-valued interpretation* we understand any four-valued interpretation I which never assumes the truth value C; i.e. an interpretation $I = \langle I^+, I^- \rangle$ such that $I^+ \cap I^- = \emptyset$. If I is a three-valued interpretation, $I \models_3 B$ means the same as $I \models_4 B$, the subscript 3 is used just to emphasize that I is three-valued; we will often write just $I \models B$. Given a set of propositional formulas $\Gamma \cup \{B\}$, we write $\Gamma \models_3 B$ to represent that every three-valued model for Γ is a model for B.

The family of all three-valued interpretations of a fixed propositional language with the knowledge ordering $\leq_k$ is a partial order in which every non-empty subfamily has greatest lower bound, and every directed subfamily (including the empty subfamily) has least upper bound. The operator Φ_P defined in the four-valued semantics, given a three-valued interpretation as an argument returns an interpretation which is also three-valued. For a general program P, the least fix-point $lfp\Phi_P$ and the greatest fix-point $gfp\Phi_P$ always exist; $lfp\Phi_P$ is always a three-valued interpretation, however $gfp\Phi_P$ can be four-valued interpretation which is not a three-valued interpretation.

The three-valued semantics of logic programs was introduced by Fitting and several of its fundamental results were established by Kunen.[10,18]

4.4. More Definitions

Recall that for a given set $Prop$ of propositional symbols, we defined $Prop^{\#} = Prop \cup \{\bar{p} \mid p \in Prop\}$. Throughout Section 4 we will use the symbol $Form$ to represent the set of all formulas with given propositional symbols and no connectives other than $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$. Also the following conventions will be used.

When we refer to an arbitrary formula of $Form(Prop)$, we will use symbol B.
When we refer to an arbitrary formula of $Form(Prop^{\#})$, we will use symbol D.
When we refer to arbitrary values in $\{T, F, U, C\}$, we will use symbols x, y.
When we refer to an arbitrary value in $\{0, 1\}$, we will use symbol d.
When we refer to a two-valued interpretation, we will use symbol v.
When we refer to a four- or three-valued interpretation, we will use symbol I.
These symbols will be also used with primes or with indices.

Definition 52 (Positive transformation and negative transformation)
We define the *positive transformation* $(\cdot)^+$ and *negative transformation* $(\cdot)^-$:

$(\cdot)^+ : Form(Prop) \rightarrow Form(Prop^{\#})$	$(\cdot)^- : Form(Prop) \rightarrow Form(Prop^{\#})$
$p^+ = p$	$p^- = \bar{p}$
$\top^+ = \top$	$\top^- = \bot$
$\bot^+ = \bot$	$\bot^- = \top$
$(B_1 \wedge B_2)^+ = B_1^+ \wedge B_2^+$	$(B_1 \wedge B_2)^- = B_1^- \vee B_2^-$
$(B_1 \vee B_2)^+ = B_1^+ \vee B_2^+$	$(B_1 \vee B_2)^- = B_1^- \wedge B_2^-$
$(\neg B)^+ = B^-$	$(\neg B)^- = B^+$
$(B_1 \equiv B_2)^+ =$	$(B_1 \equiv B_2)^- =$
$\quad (B_1^+ \equiv B_2^+) \wedge (B_1^- \equiv B_2^-)$	$\quad \neg((B_1^+ \equiv B_2^+) \wedge (B_1^- \equiv B_2^-))$
$(B_1 \leftarrow B_2)^+ =$	$(B_1 \leftarrow B_2)^- =$
$\quad (B_1^+ \leftarrow B_2^+) \wedge (B_1^- \leftarrow B_2^-)$	$\quad \neg((B_1^+ \leftarrow B_2^+) \wedge (B_1^- \leftarrow B_2^-))$

We will assume that $^+$ and $^-$ bind stronger than any connective, (so e.g. $\neg B^-$

means $\neg(B^-)$, not $(\neg B)^-$.) □

This definition extends positive and negative transformations defined in Subsection 3.1 to formulas which include also $\equiv, \leftarrow$. This definition will be crucial for describing the correspondence between multivalued semantics and the two-valued semantics as well as for analyzing internal symmetries of the constructive completion.

The following remark relates the positive transformation to the constructive transformation of a program.

Remark 53 Earlier in the paper we used the symbol P^+ to denote the constructive transformation of general program P. We will argue here that this use of the symbol $^+$ is consistent (up to the classical or intuitionistic equivalence) with the use of $^+$ to represent the positive transformation $^+ : Form(Prop^\#) \to Form(Prop^\#)$.

Let $p \leftarrow B$ be the unique statement in P^C which defines p. While building the constructive transformation P^+, this statement is replaced by the statements $(p^+ \leftarrow B^+)$ and $(p^- \leftarrow B^-)$. The conjunction of these statements is equivalent in intuitionistic logic to $(p \leftarrow B)^+$. We can extend positive transformation $(\cdot)^+$ to sets of formulas: $\Gamma^+ = \{B^+ \mid B \in \Gamma\}$. With this extension, P^+ can be read as representing either a constructive transformation of P or the positive transformation of P, but there is no inconsistency: these two sets of formulas are equivalent in intuitionistic logic, classical logic and any intermediate logic. □

Remark 54 We will compare $Comp(P^+)$ and $(Comp(P))^+$. In the construction of constructive completion $Comp(P^+)$ the reversed implications $\leftarrow$ in the statements of P^+ are replaced by equivalences $\equiv$. Thus the formulas in $Comp(P^+)$ which correspond to the statement $p \leftarrow B$ of P^C are $p^+ \equiv B^+$ and $p^- \equiv B^-$. The conjunction of these formulas is equivalent in intuitionistic logic to $(p \equiv B)^+$. So, $Comp(P^+)$ and $(Comp(P))^+$ are equivalent in intuitionistic logic, classical logic and any intermediate logic. □

The remaining part of this subsection contains definitions and lemmas concerned with notation which allows us to handle the cases of different connectives simultaneously. Additional lemmas concern symmetries in the definitions of connectives. This material is technical in nature and will have no bearing on the further presentation of the theory, however it will be used in some proofs.

Definition 55 For any operation $* : \{T, F, U, C\}^n \to \{T, F, U, C\}$, we define:
$*^+ : \{0,1\}^{2n} \to \{0,1\}$ as $*^+(d_1, d_1', ..., d_n, d_n') = (*(\langle d_1, d_1'\rangle, ..., \langle d_n, d_n'\rangle))^+$, and
$*^- : \{0,1\}^{2n} \to \{0,1\}$ as $*^-(d_1, d_1', ..., d_n, d_n') = (*(\langle d_1, d_1'\rangle, ..., \langle d_n, d_n'\rangle))^-$. □

Example 56

$$\wedge^+(d_1, d_1', d_2, d_2') = (\langle d_1, d_1'\rangle \wedge \langle d_2, d_2'\rangle)^+ = \langle d_1 \wedge d_2, d_1' \vee d_2'\rangle^+ = d_1 \wedge d_2,$$
$$\wedge^-(d_1, d_1', d_2, d_2') = (\langle d_1, d_1'\rangle \wedge \langle d_2, d_2'\rangle)^- = \langle d_1 \wedge d_2, d_1' \vee d_2'\rangle^- = d_1' \vee d_2',$$
$$\neg^+(d_1, d_1') = (\neg\langle d_1, d_1'\rangle)^+ = \langle d_1', d_1\rangle^+ = d_1',$$
$$\neg^-(d_1, d_1') = (\neg\langle d_1, d_1'\rangle)^- = \langle d_1', d_1\rangle^- = d_1.$$
□

Definition 57 For every connective $*$ among $\top, \bot, \wedge, \vee, \neg$ we define functions:

$*^+ : (Form(Prop^\#))^{2n} \to Form(Prop^\#)$ $\quad *^- : (Form(Prop^\#))^{2n} \to Form(Prop^\#)$
where n is the arity of $*$:

$$
\begin{array}{ll}
\top^+ = \top & \top^- = \bot \\
\bot^+ = \bot & \bot^- = \top \\
\wedge^+(D_1, D'_1, D_2, D'_2) = D_1 \wedge D_2 & \wedge^-(D_1, D'_1, D_2, D'_2) = D'_1 \vee D'_2 \\
\vee^+(D_1, D'_1, D_2, D'_2) = D_1 \vee D_2 & \vee^-(D_1, D'_1, D_2, D'_2) = D'_1 \wedge D'_2 \\
\neg^+(D_1, D'_1) = D'_1 & \neg^-(D_1, D'_1) = D_1 \\
\equiv^+ (D_1, D'_1, D_2, D'_2) = & \equiv^- (D_1, D'_1, D_2, D'_2) = \\
\quad (D_1 \equiv D_2) \wedge (D'_1 \equiv D'_2) & \quad \neg((D_1 \equiv D_2) \wedge (D'_1 \equiv D'_2)) \\
\leftarrow^+ (D_1, D'_1, D_2, D'_2) = & \leftarrow^- (D_1, D'_1, D_2, D'_2) = \\
\quad (D_1 \leftarrow D_2) \wedge (D'_1 \leftarrow D'_2) & \quad \neg((D_1 \leftarrow D_2) \wedge (D'_1 \leftarrow D'_2)) \quad \Box
\end{array}
$$

The functions $*^+$ and $*^-$ were defined so that the following lemma could be obtained.

Lemma 58 *For every symbol $*$ among $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$ and for all formulas $B_1, ..., B_n$ over Prop, where $n \geq 0$ is the arity of $*$, the following equalities hold:*

$$(*(B_1, ..., B_n))^+ = *^+(B_1^+, B_1^-, B_2^+, B_2^-),$$
$$(*(B_1, ..., B_n))^- = *^-(B_1^+, B_1^-, B_2^+, B_2^-).$$

Proof. For $\wedge$ we have:

$$(B_1 \wedge B_2)^+ = B_1^+ \wedge B_2^+ = \wedge^+(B_1^+, B_1^-, B_2^+, B_2^-),$$
$$(B_1 \wedge B_2)^- = B_1^- \vee B_2^- = \wedge^-(B_1^+, B_1^-, B_2^+, B_2^-).$$

For $\neg$ we have:

$$(\neg B_1)^+ = B_1^- = \neg^+(B_1^+, B_1^-),$$
$$(\neg B_1)^- = B_1^+ = \neg^-(B_1^+, B_1^-).$$

The reader can verify cases of other connectives. $\Box$

The following lemma formalizes the observation that the expression such as $\wedge^+(d_1, d'_1, d_2, d'_2) = d_1 \wedge d_2$ defining $\wedge^+ : \{0,1\}^4 \to \{0,1\}$ and the expression $\wedge^+(D_1, D'_1, D_2, D'_2) = D_1 \wedge D_2$ defining $\wedge^+ : Form(Prop^\#)^4 \to Form(Prop^\#)^4$ look alike.

Lemma 59 *For every symbol $*$ among $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$, for all formulas $B_1, ..., B_n$ over Prop, where $n \geq 0$ is the arity of $*$, and for every two-valued interpretation v of $Prop^+$ the following equalities hold:*

$$v(*^+(B_1^+, B_1^-, ..., B_n^+, B_n^-) = *^+(v(B_1^+), v(B_1^-), ..., v(B_n^+), v(B_n^-)),$$
$$v(*^-(B_1^+, B_1^-, ..., B_n^+, B_n^-) = *^-(v(B_1^+), v(B_1^-), ..., v(B_n^+), v(B_n^-)).$$

Proof. For $\wedge$ we have:

$$
\begin{array}{l}
v(\wedge^+(B_1^+, B_1^-, B_2^+, B_2^-)) = \\
v(B_1^+ \wedge B_2^+) = \\
v(B_1^+) \wedge v(B_2^+) = \\
\langle v(B_1^+) \wedge v(B_2^+), v(B_1^-) \vee v(B_2^- \rangle^+ = \\
(\langle v(B_1^+), v(B_1^-) \rangle \wedge \langle v(B_2^+), v(B_2^-) \rangle)^+ = \\
\wedge^+(v(B_1^+), v(B_1^-), v(B_2^+), v(B_2^-))
\end{array}
$$

The equality

$$v(\wedge^-(B_1^+, B_1^-, B_2^+, B_2^-) = \wedge^-(v(B_1^+), v(B_1^-), v(B_2^+), v(B_2^-))$$

can be obtained in a similar way. For $\neg$ we have:

$$v(\neg^+(B_1^+, B_1^-)) =$$

$$v(B_1^-) =$$
$$\langle v(B_1^-), v(B_1^+)\rangle^+ =$$
$$(\neg\langle v(B_1^+), v(B_1^-)\rangle)^+ =$$
$$\neg^+(v(B_1^+), v(B_1^-))$$

The equality:

$$v(\neg^-(B_1^+, B_1^-)) = \neg^-(v(B_1^+), v(B_1^-)))$$

can be obtained in a similar way.
Cases of other connectives are similar. □

Lemma 60 *For every connective* $*$ *among* $\top, \bot, \wedge, \vee, \neg, \equiv$ *and for any* $x_1, x_1', ..., x_n, x_n' \in \{0,1\}$, *where* $n \geq 0$ *is the arity of* $*$, *the following equality holds:*

$$*^+(\neg x_1', \neg x_1, ..., \neg x_n', \neg x_n) = \neg *^- (x_1, x_1', ..., x_n, x_n').$$

Moreover this equality implies the following one:

$$*^-(\neg x_1', \neg x_1, ..., \neg x_n', \neg x_n) = \neg *^+ (x_1, x_1', ..., x_n, x_n').$$

This lemma identifies the property essential for Proposition 64. Notice that the lemma does not hold for $\leftarrow$.

Proof. For $\wedge$ we have:

$$\wedge^+(\neg x_1', \neg x_1, \neg x_2', \neg x_2) = \neg x_1' \wedge \neg x_2' = \neg(x_1' \vee x_2') = \neg \wedge^- (x_1, x_1', x_2, x_2').$$

Similarly:

$$\neg^+(\neg x_1', \neg x_1) = \neg x_1 = \neg\neg^-(x_1, x_1')$$

The reader can verify cases of other connectives. □

The following lemma identifies the property essential for Proposition 76.

Lemma 61 *For every connective* $*$ *of among* $\equiv, \leftarrow$ *and for any formulas* D_1, D_1', D_2, D_2' *the equality* $*^-(D_1, D_1', D_2, D_2') = \neg *^+ (D_1, D_1', D_2, D_2')$ *holds.*

Proof. Immediately from the definition of $*^+$ and $*^-$. □

4.5. Duality

This subsection is concerned with internal symmetries of the program P^+. These symmetries will be used later to prove results which relate multivalued semantics of P and the two-valued semantics of P^+.

Let us start by defining a natural function which transforms a two-valued interpretation of $Prop^\#$ into a two-valued interpretation of $Prop^\#$.

Definition 62
We define a function $(\cdot)^d : \{0,1\}^{Prop^\#} \to \{0,1\}^{Prop^\#}$. For any propositional symbol $p \in Prop$ we set $v^d(p) = v(\neg\bar{p})$ and $v^d(\bar{p}) = v(\neg p)$. v^d will be called a *dualization* of v. Two-valued interpretations v and v' of $Prop^\#$ are called *dual* if $v' = v^d$. □

Notice that $(v^d)^d = v$. Because of that, the notion of dual interpretations is symmetric: if v and v' are dual, so are v' and v. Also, for every v there exists a unique v', such that v and v' are dual.

The following example illustrates the relation between transformations $^+, ^-$ and dual interpretations.

Example 63 If B is $\neg p \vee (q \wedge \neg r)$ then

$$\begin{array}{llllll} B^+ & \text{is} & \bar{p} \vee (q \wedge \bar{r}), & B^- & \text{is} & p \wedge (\bar{q} \vee r), \\ \neg B^+ & \equiv & \neg\bar{p} \wedge (\neg q \vee \neg\bar{r},) & \neg B^- & \equiv & \neg p \vee (\neg\bar{q} \wedge \neg r), \end{array}$$

where the equivalences are true in classical logic. Notice that:

$$\text{if } v \models_2 B^+ \text{ then } v^d \models_2 \neg B^-,$$
$$\text{if } v \models_2 B^- \text{ then } v^d \models_2 \neg B^+.$$

This example will be generalized in the next proposition. $\square$

Proposition 64 *Let v and v' be dual two-valued interpretations of $Prop^{\#}$ and let B be any formula over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg, \equiv$. Then:*

1. $v \models_2 B^+$ *iff* $v' \models_2 \neg B^-$,
2. $v \models_2 B^-$ *iff* $v' \models_2 \neg B^+$.

Proof. We will prove points 1 and 2 in parallel, by induction.
If B is p we have:

$$v(p^+) = v(p) = v'(\neg\bar{p}) = v'(\neg p^-), \text{ and}$$
$$v(p^-) = v(\bar{p}) = v'(\neg p) = v'(\neg p^+).$$

The inductive case relies on Lemmas 58, 59, 60. If B is $*(B_1, ..., B_n)$, where $*$ is one of the connectives $\top, \bot, \wedge, \vee, \neg, \equiv$ used in a prefix way, and $n \geq 0$ is the arity of $*$, then

$$\begin{array}{l} v((*(B_1, ..., B_n))^+) = \\ v(*^+(B_1^+, B_1^-, ..., B_n^+, B_n^-) = \\ *^+(v(B_1^+), v(B_1^-), ..., v(B_n^+), v(B_n^-)) = \\ *^+(v'(\neg B_1^-), v'(\neg B_1^+), ..., v'(\neg B_n^-), v'(\neg B_n^+)) = \\ *^+(\neg v'(B_1^-), \neg v'(B_1^+), ..., \neg v'(B_n^-), \neg v'(B_n^+)) = \\ \neg *^-(v'(B_1^+), v'(B_1^-), ..., v'(B_n^+), v'(B_n^-)) = \\ v'(\neg *^-(B_1^+, B_1^-, ..., B_n^+, B_n^-) = \\ v'(\neg * (B_1, ..., B_n)^-), \end{array}$$

and

$$\begin{array}{l} v((*(B_1, ..., B_n))^-) = \\ v(*^-(B_1^+, B_1^-, ..., B_n^+, B_n^-) = \\ *^-(v(B_1^+), v(B_1^-), ..., v(B_n^+), v(B_n^-)) = \\ *^-(v'(\neg B_1^-), v'(\neg B_1^+), ..., v'(\neg B_n^-), v'(\neg B_n^+)) = \\ *^-(\neg v'(B_1^-), \neg v'(B_1^+), ..., \neg v'(B_n^-), \neg v'(B_n^+)) = \\ \neg *^+(v'(B_1^+), v'(B_1^-), ..., v'(B_n^+), v'(B_n^-)) = \\ v'(\neg *^+(B_1^+, B_1^-, ..., B_n^+, B_n^-) = \\ v'(\neg * (B_1, ..., B_n)^+). \end{array}$$

$\square$

Proposition 65 *Let v and v' be dual two-valued interpretations of $Prop^{\#}$ and let B_1, B_2 be any formulas over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$. Then:*

1. $v \models_2 (B_1 \equiv B_2)^+$ *iff* $v' \models_2 (B_1 \equiv B_2)^+$,
2. $v \models_2 (B_1 \equiv B_2)^-$ *iff* $v' \models_2 (B_1 \equiv B_2)^-$,
3. $v \models_2 (B_1 \leftarrow B_2)^+$ *iff* $v' \models_2 (B_2 \leftarrow B_1)^+$,
4. $v \models_2 (B_1 \leftarrow B_2)^-$ *iff* $v' \models_2 (B_2 \leftarrow B_1)^-$.

Proof. For a proof of point 3, consider the following:

$$\begin{array}{l} v((B_1 \leftarrow B_2)^+) = \\ v((B_1^+ \leftarrow B_2^+) \wedge (B_1^- \leftarrow B_2^-)) = \end{array}$$

$$
\begin{aligned}
&(v(B_1^+) \leftarrow v(B_2^+)) \wedge (v(B_1^-) \leftarrow v(B_2^-)) = \\
&(v'(\neg B_1^-) \leftarrow v'(\neg B_2^-)) \wedge (v'(\neg B_1^+) \leftarrow v'(\neg B_2^+)) = \\
&(\neg v'(B_1^-) \leftarrow \neg v'(B_2^-)) \wedge (\neg v'(B_1^+) \leftarrow \neg v'(B_2^+)) = \\
&(v'(B_2^-) \leftarrow v'(B_1^-)) \wedge (v'(B_2^+) \leftarrow v'(B_1^+)) = \\
&(v'(B_2^+) \leftarrow v'(B_1^+)) \wedge (v'(B_2^-) \leftarrow v'(B_1^-)) = \\
&v'((B_2^+ \leftarrow B_1^+) \wedge (B_2^- \leftarrow B_1^-)) = \\
&v'((B_2 \leftarrow B_1)^+).
\end{aligned}
$$

For a proof of point 4, consider the following:

$$
\begin{aligned}
&v((B_1 \leftarrow B_2)^-) = \\
&v(\neg((B_1^+ \leftarrow B_2^+) \wedge (B_1^- \leftarrow B_2^-))) = \\
&\neg(v(B_1^+) \leftarrow v(B_2^+)) \wedge (v(B_1^-) \leftarrow v(B_2^-)) = \\
&\neg(v'(\neg B_1^-) \leftarrow v'(\neg B_2^-)) \wedge (v'(\neg B_1^+) \leftarrow v'(\neg B_2^+)) = \\
&\neg(\neg v'(B_1^-) \leftarrow \neg v'(B_2^-)) \wedge (\neg v'(B_1^+) \leftarrow \neg v'(B_2^+)) = \\
&\neg(\neg v'(B_1^+) \leftarrow \neg v'(B_2^+)) \wedge (\neg v'(B_1^-) \leftarrow \neg v'(B_2^-)) = \\
&\neg(v'(B_2^+) \leftarrow v'(B_1^+)) \wedge (v'(B_2^-) \leftarrow v'(B_1^-)) = \\
&v'(\neg((B_2^+ \leftarrow B_1^+) \wedge (B_2^- \leftarrow B_1^-))) = \\
&v'((B_2 \leftarrow B_1)^-).
\end{aligned}
$$

Proofs of points 1 and 2 are analogous to those of 3 and 4. □

The next definition extends Boolean operations $\wedge, \vee, \neg$ to two-valued interpretations by a pointwise application. It also introduces infinite counterparts of $\wedge, \vee$.

Definition 66

We define a operations $\cap, \cup, -, \bigcap, \bigcup$ on two-valued interpretations of $\mathit{Prop}^\#$. For any propositional symbol $A \in \mathit{Prop}^\#$ we set

$(v_1 \cap v_2)(A) = v_1(A) \wedge v_2(A)$,

$(v_1 \cup v_2)(A) = v_1(A) \vee v_2(A)$,

$(-v)(A) = \neg v(A)$,

$(\bigcap_{k\in K} v_k)(A) = 1$ if for every $j \in J$, $v_j(A) = 1$, otherwise $(\bigcap_{k\in K} v_k)(A) = 0$,

$(\bigcup_{k\in K} v_k)(A) = 1$ if there is $j \in J$ s.t. $v_j(A) = 1$, otherwise $(\bigcup_{k\in K} v_k)(A) = 0$. □

Remark 67 Although the operations on two-valued interpretations above are defined by a pointwise application of Boolean operations $\wedge, \vee, \neg$, we denote them by set-theoretical symbols $\cap, \cup, -$, not by $\wedge, \vee, \neg$ in order to avoid a possible confusion. Notice that the equality $(-v)(D) = v(\neg D)$ holds only propositional symbols, but not for complex formulas D. Similarly, for complex formulas, $(v_1 \cap v_2)(D)$ differs from $v_1(D) \wedge v_2(D)$, and $(v_1 \cup v_2)(D)$ differs from $v_1(D) \vee v_2(D)$.

Using set theoretical symbols in this context becomes natural if we think of a two-valued interpretation v of $\mathit{Prop}^\#$ as representing the set $\{A \in \mathit{Prop}^\# \mid v(A) = 1\}$. Then, $-v$ represents the complement of the set represented by v, $v_1 \cap v_2$ represents the intersection of sets represented by v_1, v_2, and similarly for $\cup, \bigcap, \bigcup$

Notice that relation $v_1 \leq v_2$ holds if the set represented by v_1 is included in the set represented by v_2. Because of that, it is obvious that

$$
\begin{aligned}
&v_1 \cap v_2 = \mathit{glb}\{v_1, v_2\}, \\
&v_1 \cup v_2 = \mathit{lub}\{v_1, v_2\}, \\
&\textstyle\bigcap_{k\in K} v_k = \mathit{glb}\{v_k \mid k \in K\}, \\
&\textstyle\bigcup_{k\in K} v_k = \mathit{lub}\{v_k \mid k \in K\}.
\end{aligned}
$$

□

Proposition 68 *Let P be a general program over Prop. Then, the function $(\cdot)^d$ is an isomorphism from the structure*[b]

$$\langle \{0,1\}^{Prop^\#}, \leq, \cap, \bigcap, \cup, \bigcup, -, {}^d, T_{P+} \rangle$$

onto the structure

$$\langle \{0,1\}^{Prop^\#}, \geq, \cup, \bigcup, \cap, \bigcap, -, {}^d, T_{P+} \rangle.$$

Proof. To show that ^{d} is an isomorphism we need to verify that:

1. $(\cdot)^d$ is 1-1 and onto,
2. $v_1 \leq v_2$ iff $v_1^d \geq v_2^d$,
3. $(v_1 \cap v_2)^d = v_1^d \cup v_2^d$,
4. $(\bigcap_{k \in K} v_k)^d = \bigcup_{k \in K} v_k^d$,
5. $(v_1 \cup v_2)^d = v_1^d \cap v_2^d$,
6. $(\bigcup_{k \in K} v_k)^d = \bigcap_{k \in K} v_k^d$,
7. $(-v)^d = -(v^d)$,
8. $(v^d)^d = (v^d)^d$,
9. $(T_{P+}(v))^d = T_{P+}(v^d)$.

Point 1 follows from the observation that $(v^d)^d = v$.

For a proof of point 2 consider:

$v_1 \leq v_2$, iff
for every $p \in Prop$, $v_1(p) \leq v_2(p)$ and $v_1(\bar{p}) \leq v_2(\bar{p})$, iff
for every $p \in Prop$, $\neg v_1(p) \geq \neg v_2(p)$ and $\neg v_1(\bar{p}) \geq \neg v_2(\bar{p})$, iff
for every $p \in Prop$, $v_1^d(\bar{p}) \leq v_2^d(\bar{p})$ and $v_1^d(p) \leq v_2^d(p)$, iff
$v_1^d \geq v_2^d$.

Point 3 follows from point 2, because $\cap$ can be defined in terms of $\leq$ by a first-order formula. (It is also straightforward to verify point 3 directly.)

Point 4 follows from points 1 and 2 because $\bigcap_{k \in K} v_k = glb\{v_k \mid k \in K\}$. (It is also straightforward to verify this equality directly.)

Point 5 is analogous to point 3.

Point 6 is analogous to point 4.

For a proof of point 7 consider:

$$((-v)^d)(p) = \neg(v^d(p)) = \neg\neg v(\bar{p}) = (-v)^d(p),$$
$$((-v)^d)(\bar{p}) = \neg(v^d(\bar{p})) = \neg\neg v(p) = (-v)^d(\bar{p}).$$

Point 8 is trivially true.

For a proof of point 9 assume that $p \leftarrow B$ is the unique statement in P^C which defines p. By Proposition 64 we have:

$$T_{P+}(v^d)(p) = v^d(B^+) = v(\neg B^-) = T_{P+}(v)(\neg\bar{p}) = (T_{P+}(v))^d(p),$$
$$T_{P+}(v^d)(\bar{p}) = v^d(B^-) = v(\neg B^+) = T_{P+}(v)(\neg p) = (T_{P+}(v))^d(\bar{p}). \qquad \square$$

As a corollary to Proposition 68 we obtain:

Proposition 69 *For any general program P:*

1. If v and v' are dual then $T_{P+}(v)$ and $T_{P+}(v')$ are dual,
2. $T_{P+} \uparrow \alpha$ and $T_{P+} \downarrow \alpha$ are dual, for any ordinal α,
3. lfpT_{P+} and gfpT_{P+} are dual,
4. If v and v' are dual and v is a fix-point of T_{P+} then v' is a fix-point of T_{P+}.

[b] We consider a generalization of the notion of relational structure in which infinite operations are allowed.

Proof.
1. If v and v' are dual, $v' = v^d$ and $T_{P^+}(v') = T_{P^+}(v^d) = (T_{P^+}(v))^d$.
2. Follows immediately from Proposition 68.
3. Follows immediately from Proposition 68.
4. If v and v' are dual and $T_{P^+}(v) = v$ then
$T_{P^+}(v') = T_{P^+}(v^d) = (T_{P^+}(v))^d = v^d = v'$. □

Immediately from Propositions 64 and 69 we obtain:

Corollary 70 *Let P be a general program and B a formula over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg, \equiv$. Then*

1. *$lfpT_{P^+} \models_2 B^+$ iff $gfpT_{P^+} \models_2 \neg B^-$,*
2. *$lfpT_{P^+} \models_2 B^-$ iff $gfpT_{P^+} \models_2 \neg B^+$.*

The properties listed in this corollary can be compared with those in the following lemma.

Lemma 71 *Let P be a general program and B a formula over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg$. Then:*

1. *If $lfpT_{P^+} \models_2 B^+$ then $lfpT_{P^+} \models_2 \neg B^-$,*
2. *If $lfpT_{P^+} \models_2 B^-$ then $lfpT_{P^+} \models_2 \neg B^+$,*
3. *If $gfpT_{P^+} \models_2 \neg B^+$ then $gfpT_{P^+} \models_2 B^-$,*
4. *If $gfpT_{P^+} \models_2 \neg B^-$ then $gfpT_{P^+} \models_2 B^+$.*

Proof. If $lfpT_{P^+} \models_2 B^+$ then by Propositions 69 and 64, $gfpT_{P^+} \models_2 \neg B^-$. As B^- contains no connectives other than $\top, \bot, \wedge, \vee$, by the monotonicity of B^-, $lfpT_{P^+} \models_2 \neg B$. This proves point 1. Proofs of points 2-4 are analogous. □

Proposition 72 *Let P be a general program over $Prop^\#$ and let v and v' be dual interpretations of $Prop^\#$. Then: $v \models_2 Comp(P^+)$ iff $v' \models_2 Comp(P^+)$.*

Notice however that the following is not true for dual v and v':

$$v \models_2 P^+ \text{ iff } v' \models_2 P^+.$$

Proof. From Proposition 69, point 4, and the fact that fix-points of T_{P^+} coincide with models of $Comp(P^+)$. □

4.6. Comparison of Fix-point Semantics

In this subsection we will study a correspondence between multivalued interpretations of *Prop* and two-valued interpretations of $Prop^\#$. We will show that four-valued fix-point semantics of P is equivalent to two-valued fix-point semantics of P^+.

We will define a mapping from two-valued interpretations of $Prop^\#$ to four-valued interpretations of *Prop*.

Definition 73 We define a function $c : \{0,1\}^{Prop^\#} \rightarrow \{T, F, U, C\}^{Prop}$. For a two-valued interpretation v of $Prop^\#$ for every $p \in Prop$,

$$c(v)(p) = \langle v(p), v(\bar{p})\rangle.$$

A four-valued interpretation I of *Prop* and two-valued interpretation v of $Prop^{\#}$ will be called *corresponding interpretations* if $I = c(v)$. □

In the following proposition we will list equivalent characterizations of the notion of corresponding interpretations. The second and the third of these characterizations could be used to define a mapping inverse to c.

Proposition 74 *Let I be a four-valued interpretation of Prop. Let v be a two-valued interpretation of $\mathrm{Prop}^{\#}$. Then the following conditions are equivalent:*

1. *For every $p \in \mathrm{Prop}$, $I(p) = \langle v(p), v(\bar{p})\rangle$,*
2. *For every $p \in \mathrm{Prop}$, $v(p) = (I(p))^{+}$ and $v(\bar{p}) = (I(p))^{-}$,*
3. *For every $p \in \mathrm{Prop}$, $v(p) = \chi_{I^{+}}(p)$ and $v(\bar{p}) = \chi_{I^{-}}(p)$,*

(where χ_S is the characteristic function of the set S).

Proof. Straightforward. □

Proposition 75 [c] *Let I be a four-valued (or three-valued) interpretation of Prop, and let v be the corresponding two-valued interpretation of $\mathrm{Prop}^{\#}$. Let B be any formula over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$. Then:*

1. *$I(B) = \langle v(B^{+}), v(B^{-})\rangle$,*
2. *$I \models B$ iff $v \models_2 B^{+}$ and $v \models_2 \neg B^{-}$.*

Proof. We prove point 1 by induction on B.
If B is a propositional symbol p:
$I(p) = \langle \chi_I^{+}(p), \chi_I^{-}(p)\rangle = \langle v(p), v(\bar{p})\rangle = \langle v(p^{+}), v(p^{-})\rangle$.
If B is $*(B_1, ..., B_n)$, where $*$ is one of the connectives $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$ used in a prefix way, and n is the arity of $*$, then by Proposition 75:
$I(*(B_1, ..., B_n)) =$
$*(I(B_1), ..., I(B_n)) =$
$*(\langle v(B_1^{+}), v(B_1^{-})\rangle, ..., \langle v(B_n^{+}), v(B_n^{-})\rangle) =$
$\langle *^{+}(v(B_1^{+}), v(B_1^{-}), ..., v(B_n^{+}), v(B_n^{-})), *^{-}(v(B_1^{+}), v(B_1^{-}), ..., v(B_n^{+}), v(B_n^{-}))\rangle =$
$\langle v(*^{+}(B_1^{+}, B_1^{-}, ..., B_n^{+}, B_n^{-})), v(*^{-}(B_1^{+}, B_1^{-}, ..., B_n^{+}, B_n^{-}))\rangle =$
$\langle v((*(B_1, ..., B_n))^{+}), v((*(B_1, ..., B_n))^{-})\rangle$.
Point 2 follows from point 1. □

Proposition 76 *Let I be a four-valued (or three-valued) interpretation of Prop, and let v be the corresponding two-valued interpretation of $\mathrm{Prop}^{\#}$. Let B_1, B_2 be any formulas over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$. Then:*

1. *$I \models B_1 \equiv B_2$ iff $v \models_2 B_1^{+} \equiv B_2^{+}$ and $v \models_2 B_1^{-} \equiv B_2^{-}$,*
2. *$I \models B_1 \leftarrow B_2$ iff $v \models_2 B_1^{+} \leftarrow B_2^{+}$ and $v \models_2 B_1^{-} \leftarrow B_2^{-}$.*

Proof. For a proof of point 1 consider the following:
$I \models B_1 \equiv B_2$, iff
$I(B_1 \equiv B_2) = T$, iff (by Proposition 75)
$v((B_1 \equiv B_2)^{+}) = 1$ and $v((B_1 \equiv B_2)^{-}) = 0$, iff (by Lemma 61)
$v((B_1 \equiv B_2)^{+}) = 1$ and $v(\neg(B_1 \equiv B_2)^{+}) = 0$, iff
$v((B_1 \equiv B_2)^{+}) = 1$ and $v((B_1 \equiv B_2)^{+}) = 1$, iff

[c] This proposition is a strengthening of Lemma 3.3 of Stärk, allowing $\equiv, \leftarrow$ to occur in the formula.[28]

$v((B_1 \equiv B_2)^+) = 1$, iff (by definition of $^+$)
$v((B_1^+ \equiv B_2^+) \wedge (B_1^- \equiv B_2^-)) = 1$, iff
$v(B_1^+ \equiv B_2^+) = 1$ and $v(B_1^- \equiv B_2^-) = 1$, iff
$v \models_2 B_1^+ \equiv B_2^+$ and $v \models_2 B_1^- \equiv B_2^-$.
The proof of point 2 is analogous. $\square$

Definition 77
The family of all three-valued interpretations of $Prop$ will be denoted by $S_3(Prop)$. We will call a two-valued interpretation v of $Prop^\#$ *consistent* if for no $p \in Prop$ both $v(p) = 1$ and $v(\bar{p}) = 1$. The family of all consistent two-valued interpretations of $Prop$ will be denoted by $S_C(Prop^\#)$. $\square$

Definition 78
We define operations $\cap, \cup, \bigcap, \bigcup$ on four-valued interpretations:

$$\langle I_1^+, I_1^- \rangle \cap \langle I_2^+, I_2^- \rangle = \langle I_1^+ \cap I_2^+, I_1^- \cap I_2^- \rangle,$$
$$\langle I_1^+, I_1^- \rangle \cup \langle I_2^+, I_2^- \rangle = \langle I_1^+ \cup I_2^+, I_1^- \cup I_2^- \rangle,$$
$$\bigcap_{k \in K} \langle I_k^+, I_k^- \rangle = \langle \bigcap_{k \in K} I_k^+, \bigcap_{k \in K} I_k^- \rangle,$$
$$\bigcup_{k \in K} \langle I_k^+, I_k^- \rangle = \langle \bigcup_{k \in K} I_k^+, \bigcup_{k \in K} I_k^- \rangle.$$
$\square$

Remark 79 Notice that $\langle I_1^+, I_1^- \rangle \leq_k \langle I_2^+, I_2^- \rangle$ iff $I_1^+ \subseteq I_2^+$ and $I_1^- \subseteq I_2^-$.

Notice also that

$$I_1 \cap I_2 = glb_k\{I_1, I_2\},$$
$$I_1 \cup I_2 = lub_k\{I_1, I_2\},$$
$$\bigcap_{k \in K}\{I_k\} = glb_k\{I_k \mid k \in K\},$$
$$\bigcup_{k \in K}\{I_k\} = lub_k\{I_k \mid k \in K\}.$$

Let us mention that the operations $\cap, \cup$ on four-valued interpretations can be obtained by a pointwise application of operations $\otimes, \oplus$ on $\{T, F, U, C\}$. For every $p \in Prop$,

$(I_1 \cap I_2)(p) = I_1(p) \otimes I_1(p)$ where $\langle x^+, x^- \rangle \otimes \langle y^+, y^- \rangle = \langle x^+ \wedge y^+, x^- \wedge y^- \rangle$,
$(I_1 \cup I_2)(p) = I_1(p) \oplus I_1(p)$ where $\langle x^+, x^- \rangle \oplus \langle y^+, y^- \rangle = \langle x^+ \vee y^+, x^- \vee y^- \rangle$.

The operations $\otimes$ and $\oplus$ are called the *consensus* and *accept-anything*, and were investigated by Fitting.[11] $\square$

The following theorem is central for this subsection. It expresses in a formal way the following facts:

- The four-valued fix-point semantics for general program P is an extension of the three-valued fix-point semantics for P,
- The four-valued fix-point semantics for general program P is equivalent to the two-valued fix-point semantics for positive program P^+.

Theorem 80 (Comparison of fix-point semantics)
Let P be a general program over Prop. Then c is an isomorphism from the structure

$$\langle \{0,1\}^{Prop^\#}, S_C(Prop^\#), \leq, \cap, \bigcap, \cup, \bigcup, T_{P+} \rangle$$

onto the structure

$$\langle \{T, F, U, C\}^{Prop}, S_3(Prop), \leq_k, \cap, \bigcap, \cup, \bigcup, \Phi_P \rangle.$$

Proof. To show that c is an isomorphism we need to verify that:

1. c is 1-1 and onto,

2. $v \in S_C(Prop^{\#})$ iff $c(v) \in S_3(Prop)$,
3. $v_1 \leq v_2$ iff $c(v_1) \leq_k c(v_2)$,
4. $c(v_1 \cap v_2) = c(v_1) \cap c(v_2)$,
5. $c(\bigwedge_{k \in K} v_k) = \bigcap_{k \in K} c(v_k)$,
6. $c(v_1 \cup v_2) = c(v_1) \cup c(v_2)$,
7. $c(\bigvee_{k \in K} v_k) = \bigcup_{k \in K} c(v_k)$,
8. $c(T_{P^+}(v)) = \Phi_P(c(v))$.

Point 1 can be verified using Proposition 74.
Point 2 is straightforward.
For a proof of point 3 consider:

$v_1 \leq v_2$, iff
for every $p \in Prop$, $v_1(p) \leq v_2(p)$ and $v_1(\bar{p}) \leq v_2(\bar{p})$, iff
for every $p \in Prop$, $\langle v_1(p), v_2(p)\rangle \leq_k \langle v_1(\bar{p}), v_2(\bar{p})\rangle$, iff
$c(v_1) \leq_k c(v_2)$.

Point 4 follows from point 3, because $\cap$ can be defined in terms of $\leq$ by the same first-order formula which defines $\cap$ in terms of $\leq_k$. (Alternatively, it is straightforward to verify point 4 directly.)
Point 5 follows from points 1 and 3 because $\bigcap$ is the *glb* and $\bigwedge$ is the glb_k.
Point 6 is analogous to point 4.
Point 7 is analogous to point 5.
For a proof of point 8 assume that $p \leftarrow B$ is the unique statement in P^C which defines p. By Proposition 75, for every $p \in Prop$ we have:
$c(T_{P^+}(v))(p) = \langle T_{P^+}(v)(p), T_{P^+}(v)(\bar{p})\rangle = \langle B^+, B^-\rangle = c(v)(B) = \Phi_P(c(v))(p)$ □

As a corollary we obtain:

Theorem 81 *For any general program P over Prop:*
1. *If I and v are corresponding interpretations so are $\Phi_P(I)$ and $T_{P^+}(v)$,*
2. *$\Phi_P \uparrow \alpha$ and $T_{P^+} \uparrow \alpha$ are corresponding interpretations, for any ordinal α,*
3. *$\Phi_P \downarrow \alpha$ and $T_{P^+} \downarrow \alpha$ are corresponding interpretations, for any ordinal α,*
4. *lfpΦ_P and lfpT_{P^+} are corresponding interpretations,*
5. *gfpΦ_P and gfpT_{P^+} are corresponding interpretations,*
6. *The condition "for every $p \in$ Prop, $I(p) = \langle v(p), v(\bar{p})\rangle$" defines a one-to-one correspondence between four-valued fix-points I of Φ_P and two-valued fix-points v of T_{P^+}.*

Proof. Immediately from Theorem 80. □

From Theorem 81 and Proposition 75 we obtain:

Corollary 82 *Let P be a general program and B a formula over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$. Then:*
1. *lfp$\Phi_P \models B$ iff lfp$T_{P^+} \models_2 B^+ \wedge \neg B^-$,*
2. *gfp$\Phi_P \models B$ iff gfp$T_{P^+} \models_2 B^+ \wedge \neg B^-$.*

The following proposition presents a strengthening of this corollary.

Proposition 83 *Let P be a propositional general program over Prop and B a formula over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg$. Then:*

1. $lfp\Phi_P \models B$ *iff* $lfpT_{P^+} \models_2 B^+$,
2. $gfp\Phi_P \models B$ *iff* $gfpT_{P^+} \models_2 \neg B^-$.

Proof.

$lfp\Phi_P \models B$, iff (by Theorem 81 and Proposition 75)
$lfpT_{P^+} \models_2 B^+$ and $lfpT_{P^+} \models_2 B^+$, iff (by Lemma 71)
$lfpT_{P^+} \models_2 B^+$.

This proves point 1. The proof of point 2 is analogous. □

Finally let us state how corresponding interpretations behave with respect to a program and the program's completion.

Proposition 84 *Let I be a four-valued (or three-valued) interpretation of* Prop *and let v be the corresponding two-valued interpretation of* $\mathrm{Prop}^{\#}$. *Then for any general program P over* Prop:

1. $I \models Comp(P)$ *iff* $v \models_2 Comp(P^+)$,
2. $I \models P$ *iff* $v \models_2 P^+$.

Proof. Immediately from Remark 53 and Proposition 76. □

4.7. Comparison of Declarative Semantics

In this subsection we will show that the declarative four-valued semantics of general program P, the declarative three-valued semantics of P, and the declarative two-valued semantics of P^+ are equivalent.

Let us start by showing how the four-valued logic can be reduced to the classical logic.

Theorem 85 (Belnap's four-valued logic versus classical logic)
Let $\Gamma \cup \{B\}$ be a set of formulas over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg, \equiv, \leftarrow$. *Then:*

1. $\models_4 B$ *iff* $\models_2 B^+ \wedge \neg B^-$,
2. $\Gamma \models_4 B$ *iff* $\Gamma^+ \cup neg\Gamma^- \models_2 B^+ \wedge \neg B^-$

where $\Gamma^+ = \{B_1^+ \mid B_1 \in \Gamma\}$ *and* $neg\Gamma^- = \{\neg B_1^- \mid B_1 \in \Gamma\}$.

Proof. For a proof of point 2 consider the following.
$\Gamma \models_4 B$, iff
for every I, if $I \models_4 \Gamma$ then $I \models_4 B$, iff
for every I, if for every $B_1 \in \Gamma$, $I \models_4 B_1$ then $I \models_4 B$, iff (by Proposition 75)
for every v, if for every $B_1 \in \Gamma$, $v \models_2 B_1^+ \wedge \neg B_1^-$ then $v \models_2 B^+ \wedge \neg B^-$, iff
$\Gamma^+ \cup neg\Gamma^- \models_2 B^+ \wedge \neg B^-$.

Point 1 results from point 2 by taking $\Gamma = \emptyset$. □

From Proposition 69 we obtain:

Proposition 86 *Let P be a general program and B a formula over Prop with no connectives other than* $\top, \bot, \wedge, \vee, \neg$. *Then*

$$Comp(P^+) \models_2 B^+ \text{ iff } Comp(P^+) \models_2 \neg B^-.$$

Notice however that the following are not true:

$$Comp(P^+) \models_2 B^+ \equiv \neg B^-,$$

$$P^+ \models_2 B^+ \text{ iff } P^+ \models_2 \neg B^-,$$
$$P^+ \models_2 B^+ \equiv \neg B^-.$$

Proof. Two-valued models for $Comp(P^+)$ coincide with fix-points of T_{P^+}. By this fact and monotonicity of B^+ and B^- we obtain:

$$Comp(P^+) \models_2 B^+ \text{ iff } lfpT_{P^+} \models_2 B^+,$$
$$Comp(P^+) \models_2 \neg B^- \text{ iff } gfpT_{P^+} \models_2 \neg B^-.$$

For the proof of the proposition it is enough to notice that $lfpT_{P^+} \models_2 B^+$ iff $gfpT_{P^+} \models_2 \neg B^-$, which is true by Proposition 64 because $lfpT_{P^+}$ and $gfpT_{P^+}$ are dual by Proposition 69. □

The following three theorems embody the comparison of declarative semantics. They express in a formal way that

- the four-valued declarative semantics for propositional general program P, the three-valued declarative semantics for P, and the two-valued declarative semantics for P^+ are equivalent.

Theorem 87 (Equivalence of declarative semantics) [d]
Let P be a propositional general program over Prop and B be a formula over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg$. Then the following conditions are equivalent:

1. $Comp(P) \models_4 B$,
2. $Comp(P) \models_3 B$,
3. $Comp(P^+) \models_2 B^+$.

Proof.
The implication $1 \rightarrow 2$ is obvious because every three-valued interpretation is a four valued interpretation.

To prove $2 \rightarrow 1$ assume that $Comp(P) \models_3 B$. Let I be a four-valued model for $Comp(P)$. We need to show that $I \models B$. Consider the least fix-point of Φ_P. $lfp\Phi_P$ is a three-valued interpretation and $lfp\Phi_P \models Comp(P)$, so $lfp\Phi_P \models B$. As I is a model for $Comp(P)$, it is a fix-point of Φ_P, so $lfp\Phi_P \leq_k I$. As B contains no connectives other than $\top, \bot, \wedge, \vee, \neg$, by k-monotonicity of B we obtain $I \models B$.

For a proof of $1 \leftrightarrow 3$ consider the following.

$Comp(P) \models_4 B$, iff (by Theorem 85)
$Comp(P)^+ \cup negComp(P)^- \models_2 B^+ \wedge \neg B^-$, iff
$Comp(P)^+ \models_2 B^+ \wedge \neg B^-$, iff (by Proposition 86)
$Comp(P)^+ \models_2 B^+$. □

Theorem 88 [e] *Let P be a propositional general program over Prop and B be a formula over Prop with no connectives other than $\top, \bot, \wedge, \vee, \neg$. Then:*

1. $Comp(P) \models_4 B$ *iff* $P \models_4 B$,
2. $Comp(P) \models_3 B$ *iff* $P \models_3 B$,
3. $Comp(P^+) \models_2 B^+$ *iff* $P^+ \models_2 B^+$.

[d] This theorem can be compared with Theorems 3.2 and 3.4 of Stärk, which involve a different definition of $\models_4$. Notice that 3.4 concerns formulas without $\neg$.[28]

[e] Point 2 of this theorem was proved by Stärk.[27] His proof applies also in the case of the four-valued semantics. For the sake of completeness of this presentation we are repeating his proof.

Proof.

1. For four-valued interpretations I, $I(B_1 \equiv B_2) = I((B_1 \leftarrow B_2) \wedge (B_2 \leftarrow B_1))$, so we have $Comp(P) \models_4 P$, and the implication to the left follows.
   For a proof of the implication to the right, assume that $Comp(P) \models_4 B$. Take any I such that $I \models_4 P$. We will show that $I \models_4 B$. As $I \models_4 P$, $\Phi_P(I) \leq_k I$, so there exists a fix-point I' of Φ_P such that $I' \leq_k I$. As I' is a fix-point of Φ_P, $I' \models_4 Comp(P)$, so $I' \models_4 B$, and by k-monotonicity of B, $I \models_4 B$.

2. The proof is analogous to that of point 1.

3. Notice that P^+ is a positive program and B^+ contains no connectives other than $\top, \bot, \wedge, \vee$. It is well known that the equivalence holds under such assumptions (which can be proved analogously to point 1.) □

Theorem 89 (Equivalence of declarative semantics II)
Let P be a propositional general program over Prop *and B be a formula over* Prop *with no connectives other than $\top, \bot, \wedge, \vee, \neg$. Then the following conditions are equivalent:*

1. $P \models_4 B$,
2. $P \models_3 B$,
3. $P^+ \models_2 B^+$.

Proof. Immediately from Theorems 87 and 88. □

5. Conclusion

We considered propositional logic programs with negations. For a program P we defined a constructive transformation P^+. This transformation is closely related to Mancarella's intensional negation. Using this notion we proved completeness of SLDNF-resolution with respect to deductions in classical, intuitionistic, intermediate and modal logics. The main results of this part of work are contained in Theorems 19, 40, 43 and 45. (The first two of these results were proved in the author's doctoral dissertation and announced without proofs in a conference paper.[22,23])

Then, we anlyzed the relation of the constructive transformation and multivalued semantics. In Theorem 80 we proved that the four-valued fix-point semantics for P is an extension of the three-valued fix-point semantics for P, and is equivalent to the two-valued fix-point semantics for P^+. In Theorem 87 we proved that the four-valued declarative semantics for P, the three-valued declarative semantics for P, and the two-valued declarative semantics for P^+ are equivalent. This analysis of declarative semantics is complemented by Theorem 88 whose proof idea is due to Stärk, and by Theorem 89.

We feel that it is important to revise the bulk of the theory of logic programming, replacing various complex technical notions by notions which are simpler and have more intuitive meaning. Only after such a revision can the semantics of logic programming become of practical use to programmers, and only then may those writing programs rely more on their declarative meaning than on understanding of the intricacies of the computational mechanism. The results of this paper suggest that:

- SLDNF-resolution for a program P should be replaced by SLD-resolution for the constructive transformation P^+ of this program, and
- multivalued fix-point and declarative semantics for P should be replaced by the conventional two-valued semantics for P^+.

In this paper we considered only propositional programs, however evidence exists that these two postulates are valid also for first-order programs, and for such programs they are especially important.

Acknowledgments

I would like to express my thanks to NSF for partial support of this research with grant IRI 9308970.

References

1. K. R. Apt and R. N. Bol, "Logic Programming And Negation: A Survey," *Journal of Logic Programming* vol.19/20 (1994) pp. 9-71.
2. P. Balbiani, "Modal Logic and Negation as Failure," *Journal of Logic and Computation*, vol. 1 no. 3 (1991) pp. 331-356.
3. R. Barbuti, P. Mancarella, D. Pedreschi and F. Turini, "Intensional Negation of Logic Programs: Examples and Implementation Techniques," *proceedings of International Joint Conference on Theory and Practice of Software Development TAPSOFT '87* (Springer Verlag, LNCS 250, 1987) pp. 96-110,
4. R. Barbuti, P. Mancarella, D. Pedreschi and F. Turini, "A Transformational Approach to Negation in Logic Programming," *Journal of Logic Programming* vol. 8 (1990) pp. 201-228,.
5. N. D. Belnap, Jr., "A useful four-valued logic," in: J. M. Dunn and G. Epstein (eds.), *Modern Uses of Multiple-valued Logic,* (Reidel, 1977) pp. 8-37.
6. K. L. Clark, "Negation as Failure," in *Logic and Databases*, edited by H. Gallaire and J. Minker (Plenum Press, New York, 1978) pp. 193-322.
7. K. L. Clark, "Predicate Logic as a Computational Formalism," *Research Report DOC 79/59*, (Dept. of Computing, Imperial College, 1979).
8. W. Drabent and M.Martelli, "Strict Completion of Logic Programs," *New Generation Computing* 9 (1991) pp. 69-79.
9. M. C. Fitting, *Intuitionistic Logic, Model Theory and Forcing* (North Holland, 1969).
10. M. Fitting, "A Kripke-Kleene Semantics for Logic Programs," *Journal of Logic Programming*, no. 4 (1985) pp 295-312.
11. Melvin C. Fitting, "Bilattices in the Semantics of Logic Programming," *Journal of Logic Programming*, vol. 11 (1991) pp. 91-116.
12. M. Fitting, "Basic Modal Logic," in: D.M. Gabbay, C. J. Hogger and J. A. Robinson (eds.) *Handbook of Logic in Artificial Intelligence and Logic Programming*, vol. 1, (Oxford University Press, 1993) pp. 368-448.
13. D. Gabbay, "Modal Provability Foundations for Negation as Failure I, 4^{th} draft," (February 1989) manuscript.

14. D.M. Gabbay and U. Reyle, "N-Prolog: An extension of Prolog with hypothetical implication," *Journal of Logic Programming* (1984) pp. 319-355.

15. D.M. Gabbay, C. J. Hogger and J. A. Robinson (eds.) *Handbook of Logic in Artificial Intelligence and Logic Programming* (Oxford University Press, 1993).

16. J. Jaffar, J.-L. Lassez and J. W. Lloyd, "Completeness of the negation as failure rule," in *IJCAI-83* (Karlsruhe, 1983) pp. 500-506.

17. R. A. Kowalski, "The Relation Between Logic Programming and Logic Specification," in *Mathematical Logic and Programming Languages*, edited by C. A. R. Hoare and J. C. Shepherdson (Prentice Hall, Englewood Cliffs, N.J. 1985) pp. 11-27.

18. K. Kunen, "Signed Data Dependencies in Logic Programs," *Journal of Logic Programming*, vol. 7, no. 3 (1989) pp.231-247.

19. J. W. Lloyd, *Foundations of Logic Programming*, Second extended edition (Springer Verlag, 1987).

20. P. Mancarella, D. Pedreschi, M. Rondinelli and M. Tagliatti, "Algebraic Properties of a Class of Logic Programs," in: S. Debray and M. Hermenegildo (eds.), *Logic Programming, Proceedings of the the 1990 North American Conference* (MIT Press, 1990) pp. 23-39.

21. D. Miller, "A logical Analysis of Modules in Logic Programming," *Journal of Logic Programming* vol. 6, nos. 1& 2 (1989) pp. 79-108.

22. J. A. Plaza, *Fully Declarative Programming with Logic – Mathematical Foundations* Ph.D. Dissertation, (City University of New York, July 1990).

23. J. A. Plaza, "Completeness for propositional logic programs with negation," in: Methodologies for Intelligent Systems, 6th International Symposium, edited by Z. W. Ras and M. Zemankova (Springer Verlag, LNAI 542, 1991) pp. 600-609.

24. J. A. Plaza, "Fully Declarative Logic Programming," in *Programming Language Implementation and Logic Programming, 4th International Symposium*, edited by M. Bruynooghe and M. Wirsing, (Springer Verlag, LNCS 631, 1992) pp. 415-427.

25. J. A. Plaza, "Gentzen-style Characterizations of Negation as Failure," in: J. Calmet and J. A. Campbell (eds.), *Artificial Intelligence and Symbolic Mathematical Computing, International Conference AISMC-1, Karlsruhe, Germany, August 1992, Proceedings*, (Springer Verlag, LNCS 737, 1993), pp. 237-261.

26. H. Rasiowa, *Non-classical logics, An Algebraic Approach*, (North Holland, 1974).

27. R. F. Stärk, "A complete axiomatization of the three-valued completion of logic programs," *Journal of Logic and Computation* vol. 1, no.6 (1991) pp. 811-834.

28. R. F. Stärk, "From Logic Programs to Inductive Definitions," in: W.A. Hodges (ed.), *Logic Colloquium '93*, (Oxford University Press).

29. A. Van Gelder and J. S. Schlipf, "Commonsense axiomatization for logic programs," *Journal of Logic Programming*, vol. 17, nos. 2,3,4 (1993) pp. 161-195.