A Logical Foundation for Logic Programming I: Biconsequence Relations and Nonmonotonic Completion

Alexander Bochman

e-mail: bochman@macs.biu.ac.il

Abstract

We suggest a general logical formalism for Logic Programming based on a four-valued inference. We show that it forms a proper setting for representing logic programs with negation as failure of a most general kind and for describing logics and semantics that characterize their behavior. In this way we also extend the connection between Logic and Logic Programming beyond positive programs. In addition, the suggested formalism will allow us to see a reasoning about logic programs as a most simple kind of nonmonotonic reasoning in general.

Keywords. Foundations of logic programming, negation as failure, semantics for logic programs, nonmonotonic reasoning.

1 Introduction

1.1 Logic in Logic Programming

Logic Programming was initially based on an idea that the language of classical logic can be used directly as a programming language, preserving at the same time the declarative meaning of the logical connectives. Accordingly, the rules of positive (Horn) programs can also be seen as ordinary logical formulas with the usual interpretation of the connectives involved. However, with the introduction of normal programs containing a so-called negation as failure in the bodies of their clauses, this duality between declarative (logical) and procedural interpretation has been lost, since normal program rules cannot already be interpreted as classical logical formulas. Though less immediate, an extension of the notion of a program to so-called disjunctive programs [27], and especially introduction of normal logic programs with a second, 'classical' negation [21, 1] had the same effect of weakening the connection between Logic Programming and (classical) Logic.

The absence of a clear and uniform logical basis behind logic programs with negation as failure has resulted in an unbridled proliferation of semantics suggested for such programs. In part, this was due to the fact that there was no clear ground for evaluating such semantics, apart from their computational virtues and coincidence with 'respectable' semantics in special cases. In this situation, even the existence of a semantics for all programs of a certain kind often has become a decisive reason for preferring it to semantics that lacked this feature.

In recent years, however, an important work has been done, mainly by Jürgen Dix and his colleagues, in developing some general criteria for evaluating and classifying existing semantics for logic programs – see [13, 14, 15, 16, 18]; cf. also [35]. In addition, a number of approaches have appeared that partly overcome the above problem by grounding a reasoning about logic programs and (some of) their semantics on various logical formalisms, be it three-valued logic [30], intuitionistic logic [29], a general theory of argumentation [19], or a modal logic [32].

In this study we will show that there exists a logical interpretation of program clauses of a most general kind that agrees with the majority of their procedural interpretations. In our framework, such program clauses will be considered as genuine logical formulas.

We are going to show that a logic appropriate for logic programs involving negation as failure can be based on the sole semantic principle that *truth and falsity are mutually independent notions*. According to this principle, propositions can be not only true or false, but also undetermined (neither true nor false) or contradictory (both true and false). This will quickly lead us to a four-valued semantics in which the four truth-values are identified with the subsets of the set $\{true, false\}$. This understanding of the four truth-values was suggested by Belnap in [3] and has been widely used since then, mainly in order to give representation of reasoning and logic programming in presence of both incomplete and inconsistent information (see, e.g., [4, 20, 33, 37], to mention only a few). What we are going to show here is that it can also serve as a logical basis of logic programming in general.

We will introduce a logical formalism, called *biconsequence relations*, that will give a syntactic representation for a four-valued inference based on Belnap's interpretation of the truth-values. Program clauses will be directly identified with the rules of this formalism, and the formalism itself will be considered as a formalization of the *logic* corresponding to logic programs. In this way we will restore the lost connection between Logic and Logic Programming. In the second part of this study [9], we will show that the formalism allows us to give uniform representation of various semantics for disjunctive logic programs involving negation as failure, suggested in the literature.

1.2 Logic Programming and Nonmonotonic Reasoning

The formalism of biconsequence relations will also bring us another important benefit, namely a clear separation between logical and nonmonotonic aspects of reasoning about logic programs. A most surprising general conclusion of this study, obtained on the basis of this separation, is that the distinction between various semantics for logic programs can be largely attributed to the difference in underlying (monotonic) logical systems. Moreover, most of these semantics use the same nonmonotonic principle of 'jumping to conclusions', and the only difference between them is reducible to that of the *language* used for deriving logical consequences.

The connection between logic programming and nonmonotonic reasoning is widely acknowledged these days. This connection is established mainly in the form of translation of logic programs into different nonmonotonic formalisms, such as default logic, circumscription or modal nonmonotonic logics. Investigations into the nature of this connection have turned out to be mutually profitable. For example, many of the declarative semantics suggested for logic programs have been extended to other nonmonotonic formalisms (see, e.g., [2, 24, 28]).

From the viewpoint of our framework, the connection between these two fields turns out to be even closer. In our approach, the reasoning about logic programs and their semantics constitutes a (most simple) kind of nonmonotonic reasoning in general. Thus, we show in [5] that there is a uniform way of extending the basic formalism described below to a general formalism of nonmonotonic reasoning that subsumes, among others, default logic and various modal nonmonotonic logics.

2 Biconsequence Relations and Four-Valued Inference

In this section we will describe the basic logical formalism used for representing and reasoning about logic programs. Most of the results in this and the next two sections (together with their proofs) can be found in [7] and [8].

As was said, a logical formalism we suggest for representing logic programs will be based on a four-valued entailment. More exactly, we are primarily interested in a four-valued inference based on a particular interpretation of the four truth-values suggested by Belnap in [3]. It amounts to their identification with the subsets of the set of classical truth-values $\{t, f\}$. According to this interpretation, the four truth-values $\top, \mathbf{t}, \mathbf{f}, \bot$ are identified, respectively, with $\{t, f\}$, $\{t\}$, $\{f\}$ and $\emptyset$. Accordingly, $\top$ means that a proposition is both true and false (i.e., contradictory), $\mathbf{t}$ means that it is 'classically' true (that is, true without being false), $\mathbf{f}$ means that it is classically false (without being true), while $\bot$ means that it is neither true nor false (undetermined).

The above interpretation allows us to see any 4-assignment as a *pair* of ordinary classical valuations, corresponding, respectively, to assignments of truth and falsity to propositions. To be more exact, for any 4-assignment ν (under the above interpretation) and any proposition A we can define the following two valuations saying, respectively, "A is true in ν" and "A is false in ν":

$$\begin{array}{lll} \nu \models A & \text{iff} & t \in \nu(A) \\ \nu \mathrel{=\!\!|} A & \text{iff} & f \in \nu(A) \end{array}$$

Clearly, the source 4-assignment can be restored from the above two valuations as follows:

$$\begin{array}{lll} \nu(A) = \top & \text{iff} & \nu \models A \text{ and } \nu \mathrel{=\!\!|} A \\ \nu(A) = \mathbf{t} & \text{iff} & \nu \models A \text{ and } \nu \not\mathrel{=\!\!|} A \\ \nu(A) = \mathbf{f} & \text{iff} & \nu \not\models A \text{ and } \nu \mathrel{=\!\!|} A \\ \nu(A) = \bot & \text{iff} & \nu \not\models A \text{ and } \nu \not\mathrel{=\!\!|} A \end{array}$$

The equivalence of these two representations shows that the "binary" representation is fairly general and does not restrict the set of possible four-valued interpretations. However, this representation has a significant heuristic power. In fact, it provides a natural connection, noticed already by Belnap, between four-valued reasoning and 'real' problems of commonsense reasoning and thereby the main reason for using a four-valued formalism in studying it.

2.1 Biconsequence Relations

Taking into account the representation of the four truth-values given earlier, a four-valued reasoning in general can be seen as reasoning about truth and falsity of propositions, the only distinction from classical reasoning being that the assignments of truth and falsity are independent of each other. This leads to the following construction that provides a syntactic counterpart of four-valued reasoning.

In what follows we will reserve the letters $a, b, c, \ldots$ as denoting *finite* sets of propositions, while $u, v, \ldots$ will denote arbitrary sets of propositions.

By a *bisequent* we will mean a rule of the form:

$$a : b \Vdash c : d,$$

The intended interpretation of such rules will be

"If all propositions from a are true and all propositions from b are false, then either one of the propositions from c is true or one of the propositions from d is false".

In accordance with this interpretation, propositions from a and b will be called, respectively, *positive* and *negative premises*, while those from c and d - *positive* and *negative conclusions*. As can be anticipated, bisequents will represent in what follows logic programming rules of a most general kind that can include disjunctive conclusions and default negation not only in bodies, but also in heads of the rules.

The following definition provides a primary characterization of bisequents in accordance with their intended interpretation.

Definition 2.1. A *biconsequence relation* is a set of bisequents that is closed with respect to the following rules:

(Monotonicity) $\dfrac{a : b \Vdash c : d}{a' : b' \Vdash c' : d'}$, if $a \subseteq a', b \subseteq b', c \subseteq c', d \subseteq d'$.

(Positive Reflexivity) $A : \Vdash A :$

(Negative Reflexivity) $: A \Vdash : A$

(Positive Cut) $\dfrac{a : b \Vdash A, c : d \quad A, a : b \Vdash c : d}{a : b \Vdash c : d}$

(Negative Cut) $\dfrac{a : b \Vdash c : A, d \quad a : A, b \Vdash c : d}{a : b \Vdash c : d}$

It is easy to check that the above rules agree with their informal interpretation, given above. Actually, it is easy to see that a biconsequence relation is simply a 'doubled' version of an ordinary sequent calculus. This doubling reflects the fact that truth and falsity assignments are independent of each other.

The above definition is easily extendable to arbitrary, possibly infinite sets of propositions by accepting the following *compactness requirement*:

(Compactness) $u : v \Vdash w : z$ iff $a : b \Vdash c : d$,

for some finite sets a, b, c, d such that $a \subseteq u$, $b \subseteq v$, $c \subseteq w$ and $d \subseteq z$.

In what follows we will denote by $\overline{v}$ the complement of a set v. The definition below describes the 'canonical models' of biconsequence relations. These objects play the same role as deductively closed sets (theories) for ordinary consequence relations.

Definition 2.2. A pair of sets of propositions (u, v) is a *bimodel* of a biconsequence relation $\Vdash$ if

$$u : \overline{v} \nVdash \overline{u} : v.$$

If (u, v) is a bimodel, then u will be called its *positive part*, while v the *negative part*. It is easy to show that the above condition is equivalent to the following requirement:

If $a : b \Vdash c : d$ and $a \subseteq u, b \subseteq \overline{v}$, then either $c \cap u \neq \emptyset$ or $d \cap \overline{v} \neq \emptyset$.

The requirement makes explicit in what sense bimodels can be considered as sets of propositions that are *closed* with respect to the bisequents of a biconsequence relation.

The following Representation Theorem shows that biconsequence relations are uniquely determined by their bimodels.

Representation Theorem. *If* $\Vdash$ *is a biconsequence relation, then* $a : b \Vdash c : d$ *iff, for any bimodel* (u, v)*, if* $a \subseteq u$ *and* $b \subseteq \overline{v}$*, then either* $c \cap u \neq \emptyset$ *or* $d \cap \overline{v} \neq \emptyset$*.*

Notice now that any bimodel (u, v) can be identified with a four-valued interpretation by taking its positive part u to be the set of true propositions and its negative part, v, the set of propositions that are not false. This correspondence allows us to show that biconsequence relations are adequate for their intended interpretation, namely that they provide an adequate formalization of four-valued inference (see [7] for details).

Notice that *any* set of bimodels $\mathbf{M}$ generates a biconsequence relation $\Vdash_{\mathbf{M}}$ defined as follows:

> $a : b \Vdash_{\mathbf{M}} c : d$ iff, for any bimodel $(u, v) \in \mathbf{M}$, if $a \subseteq u$ and $b \subseteq \overline{v}$, then either $c \cap u \neq \emptyset$ or $d \cap \overline{v} \neq \emptyset$.

Then the above theorem immediately implies that any biconsequence relation is determined in this sense by some set of bimodels.

2.2 Biconsequence Relations and Logic Programs

In our approach, bisequents will serve as direct representations of logic programming rules of a most general kind, namely those involving disjunction and negation in their heads (see, e.g., [23] for a use of rules with **not** in heads). To be more exact, a rule

$$\mathbf{not}\, D_1 \vee \ldots \vee \mathbf{not}\, D_k \vee C_1 \vee \ldots \vee C_l \leftarrow A_1 \wedge \ldots A_m \wedge \mathbf{not}\, B_1 \wedge \cdots \wedge \mathbf{not}\, B_n$$

will be represented by a bisequent

$$A_1, \ldots, A_m : B_1, \ldots, B_n \Vdash C_1, \ldots, C_l : D_1, \ldots, D_k.$$

Note that the formalism also allows us to represent *constraints*, that is, program rules without heads.

If S is an arbitrary set of bisequents, we will denote by $\Vdash_S$ the least biconsequence relation containing all bisequents from S. Clearly, a bisequent will belong to $\Vdash_S$ if and only if it it is provable from S using the rules of a biconsequence relation. Similarly, if S_P is a set of bisequents corresponding to program clauses of a program P, then we will denote by $\Vdash_P$ the least biconsequence relation containing S_P. This biconsequence relation can be considered as a *logical theory* corresponding to a logic program. Note also that any bimodel of $\Vdash_P$ can then be seen as a partial (Herbrand) model of P.

Using the correspondence between bisequents and program rules, we will extend in what follows the logic programming terminology to biconsequence relations. Thus, a biconsequence relation will be called *finite*, if it is generated by a finite set of bisequents, *normal*, if generated by a set of 'normal' bisequents of the form $a : b \Vdash A :$, *quasi-normal* if it is generated by normal bisequents and constraints $a : b \Vdash$, and *disjunctive* if generated by bisequents of the form $a : b \Vdash c :$. In addition, a bimodel (u, v) of a biconsequence relation will be called *consistent* if $u \subseteq v$, *complete* if $v \subseteq u$ and *classical* (total) if $u = v$.

2.3 Introducing Connectives

To begin with, note that our formalism, unlike the majority of other formalisms for many-valued reasoning, does not depend, as such, on a particular choice of four-valued connectives. Moreover, any such connective is expressible in it via introduction and elimination rules as in ordinary sequent calculi, the only distinction being that we have a pair of introduction rules and a pair of elimination rules corresponding to two premise sets and two conclusion sets, respectively (see [7, 8] for details). In this study, however, we are not interested in a four-valued reasoning in its generality. Rather, we will be primarily interested in *what information such a reasoning can give us about ordinary, classical truth and falsity*, that is about **t** and **f**. In accordance with this, our first restriction will amount to the requirement that a four-valued reasoning must agree with a classical one in cases when the context does not involve inconsistent or incomplete information. This can be secured by restricting our connectives to those that are *classical* in the sense that they give classical values when their arguments receive classical values **t** or **f**.

2.4 Classical Connectives

It turns out that in the class of all classical four-valued functions we can choose four natural connectives that are sufficient for expressing all such functions (this is shown in [8]).

The first is a well-known *disjunction* connective:

$$\nu \models A \vee B \quad \text{iff} \quad \nu \models A \text{ or } \nu \models B$$
$$\nu \mathrel{=\!|} A \vee B \quad \text{iff} \quad \nu \mathrel{=\!|} A \text{ and } \nu \mathrel{=\!|} B$$

Next, there are two unary connectives that can be seen as natural 'extensions' of a classical negation to the four-valued setting:

$$\nu \models {\sim}A \quad \text{iff} \quad \nu \mathrel{=\!|} A$$
$$\nu \mathrel{=\!|} {\sim}A \quad \text{iff} \quad \nu \models A$$

$$\nu \models \neg A \quad \text{iff} \quad \nu \not\models A$$
$$\nu \mathrel{=\!|} \neg A \quad \text{iff} \quad \nu \mathrel{\not=\!|} A$$

Note that these are the only connectives that coincide with a classical negation on the classical truth-values and satisfy the Double Negation rule. The difference between the two is that the first one 'switches' the context between truth and falsity, while the second one retains the context: ${\sim}A$ is true (false) iff A is false (resp., true), while $\neg A$ is true (false) iff A is not true (resp., not false). Accordingly, we will call $\sim$ and $\neg$ a *switching negation* and a *local negation*, respectively. Note also that each of them can be used together with the disjunction to define a natural *conjunction* connective:

$$A \wedge B \;\equiv\; {\sim}({\sim}A \vee {\sim}B),$$

or, equivalently,

$$A \wedge B \;\equiv\; \neg(\neg A \vee \neg B).$$

Finally, the following unary connective $\mathbf{L}$ can be seen as a kind of a modal operator[1]. It determines a (rudimentary) modal logic definable in the four-valued setting.

$$\nu \models \mathbf{L}A \quad \text{iff} \quad \nu \models A$$
$$\nu \mathrel{=\!|} \mathbf{L}A \quad \text{iff} \quad \nu \not\models A$$

[1]Ginsberg [22] considered L as representing the belief operator of autoepistemic logic.

It can be shown that all these connectives can be characterized using suitable introduction and elimination rules, given below. Just as in the classical case, the rules are easily discernible from the above definitions given the intended interpretation of the premises and conclusions of a bisequent.

Rules for disjunction

$$\frac{a, A : b \Vdash c : d \quad a, B : b \Vdash c : d}{a, A \vee B : b \Vdash c : d} \qquad \frac{a : b \Vdash c, A, B : d}{a : b \Vdash c, A \vee B : d}$$

$$\frac{a : b \Vdash c : d, A \quad a : b \Vdash c : d, B}{a : b \Vdash c : d, A \vee B} \qquad \frac{a : b, A, B \Vdash c : d}{a : b, A \vee B \Vdash c : d}$$

Rules for $\sim$

$$\frac{a, A : b \Vdash c : d}{a : {\sim}A, b \Vdash c : d} \qquad \frac{a : A, b \Vdash c : d}{a, {\sim}A : b \Vdash c : d}$$

$$\frac{a : b \Vdash c, A : d}{a : b \Vdash c : {\sim}A, d} \qquad \frac{a : b \Vdash c : A, d}{a : b \Vdash c, {\sim}A : d}$$

Rules for $\neg$

$$\frac{a, A : b \Vdash c : d}{a : b \Vdash \neg A, c : d} \qquad \frac{a : A, b \Vdash c : d}{a : b \Vdash c : \neg A, d}$$

$$\frac{a : b \Vdash c, A : d}{a, \neg A : b \Vdash c : d} \qquad \frac{a : b \Vdash c : A, d}{a : b, \neg A \Vdash c : d}$$

Rules for $\mathbf{L}$

$$\frac{a, A : b \Vdash c : d}{a, \mathbf{L}A : b \Vdash c : d} \qquad \frac{a : b \Vdash A, c : d}{a : \mathbf{L}A, b \Vdash c : d}$$

$$\frac{a : b \Vdash A, c : d}{a : b \Vdash \mathbf{L}A, c : d} \qquad \frac{A, a : b \Vdash c : d}{a : b, A \Vdash c : \mathbf{L}A, d}$$

Just as in the case of classical logic, the above rules can be used to show that any bisequent involving the above connectives is reducible, ultimately, to a set of bisequents containing atomic propositions only. We will call such bisequents *basic* ones. More generally, for any given language containing only 'truth-functional' four-valued connectives, there is a one-to-one correspondence between biconsequence relations and their restrictions to the basic bisequents. Note that the latter can be considered as biconsequence relations on their own right, namely as biconsequence relations in the language without connectives. Such biconsequence relations will be also called *basic*. Thus, any biconsequence relation involving only four-valued connectives is equivalent to some basic biconsequence relation.

Having the above connectives at our disposal, we can transform bisequents into more familiar rules and formulas.

For any set of propositions u, we will denote by $\sim u$ the set $\{\sim A \mid A \in u\}$. The notation $\neg u$ will have a similar meaning. In addition, for a finite set of propositions a, we will denote by $\bigvee a$ ($\bigwedge a$) the disjunction (respectively, conjunction) of all propositions from a. In the proposition below we will use also a 'material implication' $\rightarrow$ definable in the usual way from $\neg$ and $\vee$:

$$A \rightarrow B \quad \equiv \quad \neg A \vee B$$

Proposition 2.1. *Any bisequent $a : b \Vdash c : d$ is equivalent to each of the following:*

$$a, \sim b : \Vdash c, \sim d : \tag{1}$$

$$\bigwedge a \wedge \bigwedge \sim b : \Vdash \bigvee c \vee \bigvee \sim d : \tag{2}$$

$$\Vdash \bigwedge a \wedge \bigwedge \sim b \rightarrow \bigvee c \vee \bigvee \sim d : \tag{3}$$

Bisequents of the form (1) can be considered as ordinary rules with multiple consequents. Moreover, since the set of positive premises can be replaced by their conjunction and the set of positive conclusions – by their disjunction, we can transform bisequents into usual 'Tarski-type' rules (2) using only the switching negation, conjunction and disjunction. Finally, using a local negation, we can 'move' premises into conclusions, transforming thereby any bisequent into a provable formula (3). As for the classical sequent calculus, this transformation establishes an equivalence between rule-based bisequent representation and common formula-based (Hilbert-type) formalization. Moreover, it is easy to see that the disjunction and a local negation behave in an entirely classical way in this situation. In other words, any bisequent, and hence any general program clause, is representable by a logical formula in the classical language augmented with an additional negation $\sim$.

A warning note. The local negation $\neg$ should *not* be identified with the classical, or explicit, negation used in some current extensions of logic programming (see [21, 1]). The latter is usually weaker than the local negation and, in particular, does not support the above transformation (3), or even a weaker property of contraposition.

The above transformation of bisequents into logical formulas shows, in particular, that if our representation of program clauses by bisequents is justified, then the connectives involved in program clauses can indeed be considered as *logical connectives* of our four-valued logic. To be more exact, $\vee$ and $\wedge$ can be seen, respectively, as four-valued disjunction and conjunction, $\leftarrow$ corresponds in this sense to material implication $\rightarrow$, while **not** can be identified with the switching negation $\sim$. In this way, logic program clauses can, after all, be considered as logical formulas.

Remark. According to the view advocated in this work, a common understanding of **not** as 'negation by failure' (or 'negation by default') does not reflect the *meaning* of the connective, but rather a particular (nonmonotonic) mechanism of obtaining negative assumptions, a mechanism that we will describe in subsequent sections. In other words, on our view the *logical* (declarative) meaning of **not** is given by its four-valued interpretation, while the 'negation by default' principle says, roughly, that if a certain procedure of proving A fails, then we should assume **not** A 'by default' (or 'by failure', speaking in a more procedural fashion).

3 Biconsequence Relations Circumscribed

In this section we will give a description of the basic nonmonotonic construction providing a mechanism for 'jumping to conclusions' in the context of biconsequence relations. The basic idea behind the construction is the well-known *minimization principle* according to which truth and falsity of a proposition should be determined on the basis of *minimal* models satisfying a given body of knowledge or beliefs. This principle of nonmonotonic reasoning has already a long history that begins with McCarthy's Circumscription, Reiter's Closed World Assumption and Minker's Generalized CWA.

A modification of the minimization principle appropriate to our 'bicomponent' context was suggested, in fact, by Teodor Przymusinski in [31] and subsequent papers (see also [36]), though our description will be somewhat different from that given there. It amounts to a relativization of the principle with respect to negative information. As we will see in the second part of this study [9], the resulting construction actually serves as a common nonmonotonic component of all the major semantics for logic programs.

The notion of a minimal model used by the minimization principle is based on the ordering of models by inclusion with respect to propositional *atoms* that hold in them. Consequently, the definition of circumscription below is given with respect to basic biconsequence relations. Note, however, that this does not restrict the generality of our approach, since, as we said earlier, any biconsequence relation containing four-valued connectives is equivalent to some basic biconsequence relation. It should be remembered, though, that the inclusion with respect to atoms does not imply inclusion with respect to all formulas that hold in models.

A bimodel (u, v) of a basic biconsequence relation will be called *positively minimal* if there is no bimodel (u_1, v) such that $u_1 \subset u$. Now a *Positive Minimization Principle* says that only positively minimal models should matter in determining truth-values of propositions.

According to the Representation Theorem (see above), any set of bimodels determines a biconsequence relation. In particular, a set of positively minimal bimodels of any biconsequence relation $\Vdash$ determines another biconsequence relation that we will call a circumscription of $\Vdash$.

Definition 3.1. If $\mathbf{M}_m$ is a set of all positively minimal bimodels of a biconsequence relation $\Vdash$, then $\Vdash_{\mathbf{M}_m}$ will be called a *circumscription* of $\Vdash$. We will denote this biconsequence relation by $\Vdash^c$.

The following proposition (proved in [7]) provides a syntactic characterization of circumscription.

Proposition 3.1. *If* $\Vdash^c$ *is a circumscription of* $\Vdash$*, then* $a : b \Vdash^c c : d$ *iff* $: b, b' \Vdash c, c' : d, d'$*, for any* b', c', d' *such that* $: b' \Vdash c', \bigwedge a : d'$.

As can be seen, the circumscription of a biconsequence relation is uniquely determined by bisequents without positive premises that belong to it. Moreover, it is shown in [7] that $\Vdash^c$ can even be characterized as the greatest biconsequence relation having the same bisequents of such form as $\Vdash$.

Finally, note that if $\Vdash$ is represented using the corresponding logical formulas, then $\Vdash^c$ can be described as a result of circumscribing $\Vdash$ with respect to atomic propositions that are not in the scope of $\sim$ (see [31]). In other words, $\Vdash^c$ can be seen as a result of a *parameterized* circumscription that does not vary negative propositions.

Circumscription is in general a nonmonotonic operation. In other words, $\Vdash \subseteq \Vdash_1$ does not imply, in general, that $\Vdash^c$ is included in $\Vdash_1^c$. There is an important special case, however, when this feature holds.

Definition 3.2. A biconsequence relation $\Vdash_1$ will be called a *negative extension* of a biconsequence relation $\Vdash_0$ if $\Vdash_1$ coincides with the least biconsequence relation containing $\Vdash_0$ and all bisequents of the form $: b \Vdash : d$ belonging to $\Vdash_1$.

As can be shown, a negative extension is obtained by restricting possible negative parts of bimodels of the source biconsequence relation. Consequently, if $: \overline{v} \nVdash_1 : v$ holds (that is, if v is an admissible negative part of the negative extension), then any bimodel (u, v) of $\Vdash_1$ will also be a bimodel of $\Vdash_0$ and vice versa. As a result, we immediately obtain (see [7]):

Proposition 3.2. *If* $\Vdash_1$ *is a negative extension of* $\Vdash_0$*, then* $\Vdash_1^c$ *is a negative extension of* $\Vdash_0^c$*.*

This fact will be extensively used on what follows.

3.1 Circumscription and GPPE

As we said, the circumscription of a biconsequence relation is uniquely determined by bisequents without positive premises that belong to it. We will show now that this feature of the circumscribed biconsequence relation constitutes a logical basis of the *partial deduction (or evaluation) principle* for logic programs (see [10, 11, 17, 34]).

The *Generalized Principle of Partial Evaluation (GPPE)* from [10, 11, 17] can be easily formulated for biconsequence relations in general and amounts to the following:

> If S is a set of bisequents, replace a bisequent $A, a : b \Vdash c : d$ by a set of bisequents $a, a_i : b, b_i \Vdash c, c_i : d, d_i$, for every bisequent $a_i : b_i \Vdash c_i, A : d_i$ from S containing A among its positive conclusions.

As can be easily seen, GPPE can be used to eliminate bisequents containing positive premises. For example, in the case when A does not occur in heads of bisequents from S at all, any bisequent containing A among its positive premises can be simply eliminated.

GPPE has been shown to be valid for a number of semantics for logic programs. It will also turn to be valid for all the semantics generated by our general construction. This is due to the following result showing that GPPE preserves generated circumscribed biconsequence relations (see the proof at the end of the paper).

Theorem 3.3. *If S is a set of bisequents and S' is obtained from S by an application of GPPE, then $\Vdash^c_S = \Vdash^c_{S'}$.*

The semantic construction described later in this paper is based on a certain extension of the circumscribed biconsequence relation corresponding to a program. Consequently, programs generating the same circumscribed biconsequence relation will be always assigned the same semantics. In fact, GPPE can be seen as a general syntactic principle characterizing this feature of semantics for logic programs, a clear sign that a semantic is constructed, in effect, 'on top' of the circumscribed program.

In what follows, for any biconsequence relation $\Vdash$, we will denote by $R(\Vdash)$ the set of all minimal basic bisequents of the form $: b \Vdash c : d$ that belong to $\Vdash$. This set will be called a *residual* of $\Vdash$. As follows from the results stated above, the residual of a biconsequence relation uniquely determines its circumscription.

3.2 Finite Case: Circumscription and Generalized Clark Completion

In this section we will give a logical procedure for computing the circumscription of a biconsequence relation in the finite case.

For any propositional atom A, we will denote by $R^A(\Vdash)$ the set of all bisequents from the residual $R(\Vdash)$ such that $A \in c$. Then a basic biconsequence relation will be called *locally finite* if $R^A(\Vdash)$ is finite for every A in the language. As can be shown, any finite biconsequence relation is also locally finite, but not vice versa.

For any bisequent $s_i =: b_i \Vdash c_i : d_i$ from $R^A(\Vdash)$, we will denote by A_{s_i}

the following formula in the language $\{\vee, \neg, \sim\}$:

$$(A_{s_i}) \qquad \bigvee(c_i \setminus \{A\} \cup \sim\neg b_i \cup \sim d_i)$$

Notice that s_i is equivalent to $\Vdash \neg A_{s_i} \to A :$.

Let $\Vdash$ be locally finite, A a propositional atom and $s_1, \ldots, s_n$ all the bisequents from $R^A(\Vdash)$. Then we will denote by s_A the following bisequent

$$(s_A) \qquad A, A_{s_1}, \ldots, A_{s_n} : \Vdash$$

Finally, we will denote by $K(\Vdash)$ the union of $R(\Vdash)$ and all bisequents s_A, where A ranges over atomic propositions of the language. It turns out that $K(\Vdash)$ fully describes the circumscribed biconsequence relation corresponding to $\Vdash$. The proof of the next theorem can be found at the end of the paper.

Theorem 3.4. *If $\Vdash$ is a locally finite biconsequence relation, then $\Vdash^c$ coincides with $\Vdash_{K(\Vdash)}$.*

In what follows, $K(\Vdash)$ will be called a *kernel* of the circumscribed biconsequence relation.

The above theorem suggests a general method of circumscribing a (locally) finite biconsequence relation. To this end we must find first $R(\Vdash)$, that is, the set of all minimal bisequents of the form $: b \Vdash c : d$ that belong to $\Vdash$. This can be achieved by a partial deduction technique (see above) that eliminates positive premises in bisequents. After that we should add all bisequents of the form s_A. As follows from the theorem, the resulting set of bisequents will be sufficient for representing the circumscribed biconsequence relation.

Remark. The above procedure is actually equivalent to the construction of *generalized Clark's completion of the residual program* suggested in [12] for what is called there 'super logic programs'. The correspondence can be easily established once we notice that, for any A, the set of bisequents $R^A(\Vdash) \cup \{s_A\}$ is representable by the following 'completion' formula in the language $\{\vee, \neg, \sim\}$:

$$A \leftrightarrow \neg \bigwedge(\{A_{s_1}, \ldots, A_{s_n}\}),$$

where $\{s_1, \ldots, s_n\}$ is a set of all bisequents from $R^A(\Vdash)$. Consequently, the kernel of the circumscribed biconsequence relation can also be represented by a set of all such formulas.

As is easy to show, bisequents s_A are reducible to basic bisequents of the 'constraint' form $a : b \Vdash : d$. Consequently, only bisequents of the latter type need to be added to a biconsequence relation in order to produce its circumscription[2].

Corollary 3.5. *If $\Vdash$ is a locally finite biconsequence relation, then $\Vdash^c$ is the least biconsequence relation containing $R(\Vdash)$ and all bisequents of the form $a : b \Vdash : d$ from $\Vdash^c$.*

This result can be further strengthened for particular kinds of biconsequence relations. Thus, the circumscription of *disjunctive* biconsequence relations, that is, biconsequence relations determined by bisequents without negative conclusions, is determined only by bisequents of the form $a : \Vdash : d$ and $: b \Vdash c :$.

Proposition 3.6. *If $\Vdash$ is a locally finite disjunctive biconsequence relation, then $\Vdash^c$ is the least biconsequence relation containing all bisequents of the form $a : \Vdash : d$ and $: b \Vdash c :$ from $\Vdash^c$.*

The next result gives a necessary and sufficient condition for including sequents of the form $a : \Vdash : d$ into $\Vdash^c$ in this case.

Proposition 3.7. *For a disjunctive biconsequence relation $\Vdash$, $a : \Vdash^c : d$ if and only if $b \cap d \neq \emptyset$ or $: b \Vdash c :$, for any b, c such that $: b \Vdash c, \bigwedge a :$.*

If a biconsequence relation is *quasi-normal* (that is, corresponds to a normal program with constraints), then it can be shown that its circumscription is already determined by adding bisequents of the form $A : \Vdash : d$. The following results give the conditions for including such bisequents into the circumscription.

Corollary 3.8.

1. *For a quasi-normal biconsequence relation $\Vdash$, $A : \Vdash^c : d$ if and only if $: b \Vdash$ or $b \cap d \neq \emptyset$, for any b such that $: b \Vdash A :$.*

2. *If $\Vdash$ is a normal biconsequence relation, then $A : \Vdash^c : d$ if and only if $: \overline{d} \nVdash A :$.*

[2] A suitable counterexample can be produced (see [7]) showing that the finiteness restriction is essential for this result.

It can be shown that, for normal biconsequence relations, $a :\Vdash^c: d$ holds only if $A :\Vdash^c: d$, for some A from a. However, even for quasi-normal biconsequence relations this is not so, as shows the following example:

$$: C \Vdash A :$$
$$: D \Vdash B :$$
$$: C, D \Vdash$$

The corresponding circumscription contains $A, B : \Vdash$, but neither $A : \Vdash$ nor $B : \Vdash$.

4 Coherence

Belnap's interpretation of the truth-values will help us once more, this time in determining some further plausible constraints on biconsequence relations.

The main benefit of Belnap's interpretation of the four truth-values is that it allows us to use four-valued reasoning as a general framework for logical reasoning in presence of inconsistent or incomplete information. However, this generality has a weak side in that it completely ignores the distinction between ordinary truth and falsity, on the one hand, and inconsistency and incompleteness, on the other. All the four truth-values have an equal status in the context of such a reasoning. Consequently, what seems to be missing is a mechanism that would allow us to infer 'classical' information in the framework of biconsequence relations.

These considerations lead us to a natural and rather strong requirement saying that, though truth and falsity are largely independent, *provable truth and refutability must coincide with provable classical truth and falsity*. If this condition holds for a biconsequence relation, the information we can *infer* using it will be of a usual classical kind.

Biconsequence relations satisfying the above requirement will be called coherent. Note, however, that the strength of the requirement can vary depending on what propositional formulas are susceptible to coherence. Speaking generally, the more expressive is the language, the more strong is the corresponding constraint imposed by the requirement.

In what follows, by a *language* $\mathcal{L}$ we will mean a subset of classical connectives and by $\mathcal{L}$-propositions propositions constructed from atoms using connectives from $\mathcal{L}$. Then a rule that holds for a biconsequence relation will be called *logical in a language* $\mathcal{L}$ if it does not involve explicit occurrences of connectives, but is valid for substitutions of arbitrary $\mathcal{L}$-propositions for propositional atoms.

Definition 4.1. A biconsequence relation will be called *$\mathcal{L}$-coherent* if it satisfies the following two logical rules with respect to $\mathcal{L}$:

$$(\textit{Positive Coherence})\ \frac{\Vdash A :}{: A \Vdash} \qquad\qquad (\textit{Negative Coherence})\ \frac{A : \Vdash}{\Vdash : A}$$

The Positive Coherence rule says that if a proposition A is provably true, then it cannot be false (and hence it is classically true in all bimodels). Similarly, Negative Coherence says that if A cannot be true, it must be false (and hence is classically false in all bimodels).

The influence of the underlying language $\mathcal{L}$ amounts to imposing language restrictions on the applicability of the coherence rules. As we will see in the second part of the study, different semantics for logic programs can be obtained by varying these restrictions.

As is shown in [8], the actual constraint imposed by a logical rule can be 'measured' in terms of what restrictions it imposes on the associated *basic* biconsequence relation. A rule that holds for a biconsequence relation will be called *structural* if it is a logical rule with respect to the empty language $\emptyset$. In other words, a rule is structural if it does not involve connectives, but is valid for all substitutions of other propositional atoms for atoms involved in the rule. Alternatively, such a rule can be seen as a logical rule for the associated basic biconsequence relation. It turns out that logical rules considered in this paper can be always characterized in terms of some structural rules implied by it.

5 Nonmonotonic Completion and Semantics

Finally we have all we need in order to describe a nonmonotonic entailment and semantics generated by a given biconsequence relation. In fact, they arise from a combination of the two reasoning principles described earlier, namely coherence and minimization.

Definition 5.1. A *nonmonotonic $\mathcal{L}$-completion* of a biconsequence relation $\Vdash$ (denoted by ${}^{c}\Vdash^{\mathcal{L}}$) is the least $\mathcal{L}$-coherent biconsequence relation containing $\Vdash^{c}$.

Clearly, ${}^{c}\Vdash^{\mathcal{L}}$ is a biconsequence relation obtained from $\Vdash$ by first circumscribing it and then adding the coherence rules. As we will see in the second part of the study, this extension of the source biconsequence relation embodies all the information that can be nonmonotonically inferred from the latter.

'Historical' note. As a matter of fact, the above construction of the nonmonotonic completion and semantics have arisen as a certain modification of Teodor Przymusinski's earlier, non-modal construction of stationary completion and semantics for disjunctive logic programs, suggested in [31]. Actually, the very formalism used by Przymusinski in that paper was very similar to that of a four-valued logic (we will discuss this in more details in [9]). The relevant modifications made to Przymusinski's construction could be summarized as follows:

- The underlying language was allowed to vary in order to characterize alternative semantics for logic programs (in Przymusinski's construction such a variation was achieved through a change in non-monotonic principles used).

- The fixed-point construction of the stationary completion was 'splitted' into two consecutive stages: circumscription and coherence. As a result, the fixed-point characterization of the completion was replaced by coherence closure of the circumscribed consequence relation. This idea of performing circumscription 'in one step' with subsequent closure with respect to coherence rules was 'borrowed', in fact, from [36].

Let us consider the extent to which the general construction of nonmonotonic completion above depends on the underlying language. Note first that circumscription, as it is described earlier in the paper, is language independent since it is defined on the level of basic bisequents. Coherence rules, however, depend crucially on what connectives are allowed. Consequently, the influence of the language $\mathcal{L}$ amounts to imposing language restrictions on the applicability of the coherence rules. We will see, however, that in each particular case of interest there are structural rules that are equivalent to given $\mathcal{L}$-coherence rules. As a result, the description of the corresponding nonmonotonic completion can be always given without the use of the logical connectives. It should be remembered, however, that the justification of the resulting construction lies ultimately in its logical formulation.

In accordance with the above definition of a nonmonotonic completion, we can define a *nonmonotonic $\mathcal{L}$-semantics* of a biconsequence relation as a pair of sets of $\mathcal{L}$-propositions that are, respectively, provable and refutable in the nonmonotonic $\mathcal{L}$-completion:

Definition 5.2. A *nonmonotonic $\mathcal{L}$-semantics* of a biconsequence relation $\Vdash$, $SEM^{\mathcal{L}}_{\Vdash}$, is a pair $\langle SEM^{+}, SEM^{-}\rangle$ of sets of $\mathcal{L}$-propositions such that

$$SEM^{+} \;=\; \{A \mid {}^{c}{\Vdash}^{\mathcal{L}} A :\} \qquad SEM^{-} \;=\; \{A \mid A : {}^{c}{\Vdash}^{\mathcal{L}}\}$$

Due to $\mathcal{L}$-coherence, all propositions that belong to SEM^{+} will be both provably true and provably non-false in the nonmonotonic completion (that is classically true), while those from SEM^{-} will be provably classically false.

As we already said, for a general logic program P, we denote by $\Vdash_P$ the least biconsequence relation containing bisequents corresponding to all clauses from P. This biconsequence relation can be seen as representing a logical theory corresponding to P. Now, for all semantics we will consider in the second part of this study [9], P and $\Vdash_P$ (viewed as a set of program rules) will have the same semantics. In accordance with this, we can safely use SEM_P to denote $SEM_{\Vdash_P}$.

5.1 Existence

It should be clear that a biconsequence relation, even if it is consistent, may have no consistent nonmonotonic completion. A simplest example is a biconsequence relation containing both $\Vdash A :$ and $\Vdash : A$, or both $A : \Vdash$ and $: A \Vdash$, since in each of these cases the closure with respect to the coherence rules immediately produces an inconsistency. However, there is a general sufficient condition that turns out to be useful in establishing consistency of some nonmonotonic completions.

Definition 5.3. A biconsequence relation will be called *$\mathcal{L}$-regular* if it satisfies the following logical rule for $\mathcal{L}$-propositions:

$$\frac{: b \Vdash : a}{a : \Vdash b :}$$

The following proposition has been proved in [7]:

Proposition 5.1. *For any consistent biconsequence relation $\Vdash$, if $\Vdash^{c}$ is $\mathcal{L}$-regular, then $\Vdash$ has a consistent nonmonotonic $\mathcal{L}$-completion.*

A biconsequence relation will be called *affirmative* (cf. [32]) if it does not contain basic bisequents of the 'constraint' form $a : b \Vdash : d$ (in other words, any bisequent has positive conclusions). Thus, normal and disjunctive biconsequence relations are affirmative. The following proposition, also proved in [7], uses the fact that such biconsequence relations are always $\{\vee, \neg\}$-regular and, moreover, satisfy the condition of Proposition 5.1 for this language.

Proposition 5.2. *Any affirmative biconsequence relation has a consistent $\{\vee, \neg\}$-completion.*

As we will see in the second part of this study, $\{\vee, \neg\}$-completions correspond to a version of stationary semantics as well as to a natural generalization of stable class semantics for logic programs.

An important feature of nonmonotonic completions of regular biconsequence relations is that provably true and provably false propositions coincide, respectively, with propositions that are provably non-false and provably non-true. In other words, for provable propositions, truth coincides with non-falsity, and hence the nonmonotonic completion gives us, in a sense, classically coherent information.

Proposition 5.3. *Let $\{\vee, \neg\} \subseteq \mathcal{L}$ and ${}^c\Vdash$ be a nonmonotonic $\mathcal{L}$-completion of a regular biconsequence relation $\Vdash$. Then, for any $\mathcal{L}$-proposition A,*

$$\begin{array}{lcl} {}^c\Vdash\ : A & \textit{iff} & A : {}^c\Vdash \\ {}^c\Vdash A : & \textit{iff} & : A\, {}^c\Vdash \end{array}$$

6 Preliminary conclusions

In this first part of our study we have described a general logical formalism suitable for representing and reasoning about logic programs with negation by failure of a most general kind. We have presented also a uniform construction of the semantics for such programs. In the second part [9] we will show that this construction is sufficiently expressive to capture major existing semantics of logic programs of this kind.

What we see as the main advantage of our formalism is that it has allowed us to provide *a clear separation of logical and nonmonotonic aspects of reasoning about logic programs*. As a first benefit of this separation, we have 'discerned' a logic appropriate for general logic programs, and thereby restored the connection between Logic and Logic Programming. Moreover, as will be seen from the results in the second part [9], common kinds of reasoning about logic programs and their semantics are reducible, in effect, to different kinds of logical reasoning in a certain well-defined (nonmonotonic) extension of the source theory. In a sense, in this paper we have described only *one* general semantics for logic programs, more specific semantics being merely partial cases obtained by restricting the underlying language.

Note. It should be clear that the above claim is tailored to our representation of logic programs and does not exclude a possibility that some alternative formalism could achieve, in principle, the same goal of representing different semantics by preserving the underlying monotonic basis but varying, for example, its nonmonotonic part (see our discussion of Przymusinski's approach above). Cf. also [29] for still another account of the distinction between well-founded and stable semantics.

Finally, as we mentioned in the Introduction, there is a fairly general way of 'lifting' our formalism of biconsequence relations to a formalism that subsumes classical inference. In this way we will obtain such 'standard' nonmonotonic formalisms as default logic and various modal nonmonotonic logics. These issues are studied in [5]. The very possibility of such a uniform extension shows that reasoning embodied in logic programs involving negation as failure can be seen as a special case of nonmonotonic reasoning in general.

References

[1] J.J. Alferes and L.M. Pereira. (1992) On logic program semantics with two kinds of negation. In K. R. Apt, editor, *Logic Programming: Proc. 1992 Joint Int. Conf. and Symp.*, pages 574–589, Cambridge, Mass., MIT Press.

[2] C. Baral and V.S. Subrahmanian (1991) Dualities between alternative semantics for logic programming and nonmonotonic reasoning (Extended abstract). *Proc. First Int. Workshop on Logic Programming and Nonmonotonic Reasoning*, pp. 69–86.

[3] N.D. Belnap, Jr. (1977) A useful four-valued logic. In M. Dunn and G. Epstein (eds.) *Modern Uses of Multiple-Valued Logic*, D. Reidel, Dordrecht, pp. 8–41.

[4] H.A. Blair and V.S. Subrahmanian (1989) Paraconsistent logic programming. *Theoretical Computer Science*, **68**: 135–154.

[5] A. Bochman (1996) Biconsequence relations for nonmonotonic reasoning. In L.C. Aiello, J. Doyle, and S.C. Shapiro (eds.) *Principles of Knowledge Representation and Reasoning: Proc. Fifth Int. Conference (KR'96)*, Morgan Kaufmann, San Francisco, CA.

[6] A. Bochman (1996) On a logical basis of normal logic programs. *Fundamenta Informaticae*, **28**:223–245.

[7] A. Bochman (1996) What is a four-valued reasoning and how it can be nonmonotonic. (Submitted to *J. of Logic and Computation*)

[8] A. Bochman (1997) Biconsequence relations: A general formalism of reasoning with inconsistency and incompleteness. (Submitted)

[9] A. Bochman (1997) A logical foundation for logic programming II: Semantics of general logic programs. *J. of Logic Programming*

[10] S. Brass and J. Dix (1994) A disjunctive semantics based on unfolding and bottom-up evaluation. In B. Wolfinger, editor, *Innovationen bei Rechen- und Kommunikationssystemen,* (IFIP '94-Congress, Workshop FG2: Disjunctive Logic Programming and Disjunctive Databases), pages 83–91, Berlin, 1994. Springer.

[11] S. Brass and J. Dix (1995) Disjunctive semantics based upon partial and bottom-up evaluation. In L. Sterling, editor, *Proceedings of the 12th Int. Conf. on Logic Programming, Tokyo*, pages 199–213. MIT Press.

[12] S. Brass, J. Dix and T. Przymusinski (1996) Super logic programs. In L.C. Aiello, J. Doyle, and S.C. Shapiro (eds.) *Principles of Knowledge Representation and Reasoning: Proc. Fifth Int. Conference (KR'96)*, Morgan Kaufmann, San Francisco, CA., pages 529–541.

[13] J. Dix (1991) Classifying semantics of logic programs. In A. Nerode, W. Marek, and V. S. Subrahmanian, eds, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, pages 166–180, Cambridge, Mass., MIT Press.

[14] J. Dix (1992) Classifying semantics of disjunctive logic programs. In K. Apt (ed), *Proc. JICSLP'92*, the MIT Press, pp. 798–812.

[15] J. Dix (1995) A classification theory of semantics of normal logic programs: II. Weak properties. *Fundamenta Informaticae*, **22**: 257–288.

[16] J. Dix (1995) Semantics of logic programs: their intuitions and formal properties. An overview In A. Fuhrmann and H. Rott (eds.) *Logic, Action and Information – Essays on Logic in Philosophy and Artificial Intelligence*, De Gruyter, 1995, pp. 241–327.

[17] J. Dix and M. Müller (1994) Partial evaluation and relevance for approximations of the stable semantics. Z.W. Raz and M. Zemankova (eds) *Proc. ISMIS'94*, LNAI 869, pp. 511–520.

[18] J. Dix and M. Müller (1994) An axiomatic framework for representing and characterizing semantics of disjunctive logic programs. In P. Van Hentenryck, editor, *Proceedings of the 11th Int. Conf. on Logic Programming, S. Margherita Ligure*, pages 303–322. MIT Press.

[19] P.H. Dung (1995) An argumentation-theoretic foundation for logic programming. *J. of Logic Programming*, **22**: 151–177.

[20] M.C. Fitting (1989) Bilattices and the theory of truth. *J. of Philosophical Logic*, **18**: 225–256.

[21] M. Gelfond and V. Lifschitz (1991) Classical negation and disjunctive databases. *New Generation Computing*, **9**:365–385.

[22] M.L. Ginsberg (1990) Bilattices and modal operators. In *Proc. TARK'90*, ed. by R. Parikh, Morgan Kauffman, pp. 273–287.

[23] K. Inoue and C. Sakama (1994) On positive occurrences of negation as failure. In *Proc. 4th Int. Conf. on Principles of Knowledge Representation and Reasoning, KR'94*, Morgan Kauffman, San Francisco, CA., 1994, pages 293–304.

[24] V. Lifschitz (1994) Minimal belief and negation as failure. *Artificial Intelligence*, 70:53–72, 1994.

[25] V. Lifschitz and G. Schwarz (1993) Extended logic programs as autoepistemic theories. In *Proc. Second Int. Workshop on Logic Programming and Nonmonotonic Reasoning*, L.M. Pereira and A. Nerode (eds.), MIT Press, Cambridge, MA, 1993, pages 101–114.

[26] V. Lifschitz and T. Woo (1992) Answer sets in general nonmonotonic reasoning (preliminary report). In *Proc. Third Int. Conf. on Principles of Knowledge Representation and Reasoning, KR'92*, Morgan Kauffman, 1992, pages 603–614.

[27] J. Lobo, J. Minker, and A. Rajasekar (1992) *Foundations of Disjunctive Logic Programming*. MIT Press.

[28] W. Marek and V.S. Subrahmanian (1992) The relationship between stable, supported, default and autoepistemic semantics for general logic programs. *Theoretical Computer Science*, **103**: 365–386.

[29] D. Pearce (1997) A new logical characterization of stable models and answer sets. In J. Dix, L. Pereira, and T. Przymusinski, eds., *Proc. Workshop on Non-Monotonic Extensions of Logic Programming*, Springer Verlag (Lect. Notes in AI).

[30] H. Przymusinska and T. Przymusinski (1990) Semantic issues in deductive databases and logic programs, in R. Benerji (ed.) *Formal Techniques in Artificial Intelligence,* North-Holland: Amsterdam, pp. 321–367.

[31] T.C. Przymusinski (1991) Semantics of disjunctive logic programs and deductive databases. In *Proc. Second Int. Conf. on Deductive and Object-Oriented Databases, DOOD'91*, Springer Verlag, pp. 85–107.

[32] T.C. Przymusinski (1995) Static semantics for normal and disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, **14**:323–357.

[33] C. Sakama and K. Inoue (1995) Paraconsistent stable semantics for extended disjunctive programs. *J. of Automated Reasoning*, **5**: 265–285.

[34] C. Sakama and H. Seki (1994) Partial deduction of disjunctive logic programs: A declarative approach. In L. Fribourg, F. Turini (eds.) *Logic Program Synthesis and Transformation – Meta-Programming in Logic*, LNCS 883, pp. 170–181.

[35] J. S. Schlipf (1992). Formalizing a logic for logic programming. *Annals of Mathematics and Artificial Intelligence* **5**:279–302.

[36] L. Y. Yuan and J.-H. You (1993) Autoepistemic circumscription and logic programming *J. of Automated Reasoning* **10**: 143–160.

[37] G. Wagner (1994) *Vivid Logic*. Lecture Notes in AI, 764.

Proofs of the Main Results

We give here proofs for the two new theorems stated in the paper.

Theorem 3.3. *If S is a set of bisequents and S' is obtained from S by an application of GPPE, then $\Vdash^c_S = \Vdash^c_{S'}$.*

Proof. Let us assume that S' is obtained from S by replacing a bisequent $A, a_0 : b_0 \Vdash c_0 : d_0$ by a set of bisequents

$$(*) \qquad a_0, a_i : b_0, b_i \Vdash c_0, c_i : d_0, d_i,$$

for every bisequent $a_i : b_i \Vdash c_i, A : d_i$ from S containing A among its positive conclusions. We need only to show that $\Vdash_S$ and $\Vdash_{S'}$ have the same bisequents of the form $: b \Vdash c : d$.

To begin with, note that any bisequent from $(*)$ can be obtained from S by an application of the Cut rule. Consequently, $\Vdash_{S'} \subseteq \Vdash_S$. Now if $: b \nVdash_{S'} c : d$, then there is a bimodel (u, v) of $\Vdash_{S'}$ such that $b \subseteq \overline{v}$, $c \subseteq \overline{u}$ and $d \subseteq v$. If (u, v) is also a bimodel of $\Vdash_S$, then we are done, since this would imply $: b \nVdash_S c : d$. Otherwise, (u, v) should not be closed with respect to the bisequent $A, a_0 : b_0 \Vdash c_0 : d_0$ (since this is the only sequent that does not belong to $\Vdash_{S'}$). This means that $\{A\} \cup a_0 \subseteq u$, $b_0 \subseteq \overline{v}$, $c_0 \subseteq \overline{u}$ and $d_0 \subseteq v$. Since (u, v) is closed with respect to the bisequents from $(*)$, we have also that, for any bisequent $a_i : b_i \Vdash c_i, A : d_i$ from S, if $a_i \subseteq u$ and $b_i \subseteq \overline{v}$, then either $c_i \cap u \neq \emptyset$ or $d_i \cap \overline{v} \neq \emptyset$. But then it can be easily seen that $(u \setminus \{A\}, v)$ is a bimodel of $\Vdash_S$, since it is closed with respect to all bisequents from S. Consequently $: b \nVdash_S c : d$, and hence any bisequent $: b \Vdash c : d$ belongs to $\Vdash_S$ iff it belongs to $\Vdash_{S'}$. $\square$

Theorem 3.4. *If $\Vdash$ is a locally finite biconsequence relation, then $\Vdash^c$ coincides with $\Vdash_{K(\Vdash)}$.*

Proof. Notice first that bimodels of $\Vdash_{K(\Vdash)}$ are precisely pairs (u, v) of sets of propositions that are closed with respect to bisequents from $K(\Vdash)$. We will show that such bimodels coincide with positively minimal bimodels of $\Vdash$. To begin with, it is easy to show that (u, v) is closed with respect to $R(\Vdash)$ if and only if $: \overline{v} \nVdash \overline{u} : v$. This condition means that there exists a bimodel (u_0, v) of $\Vdash$ such that $u_0 \subseteq u$. In addition, (u, v) is closed with respect to all s_A iff, for any A, either $A \notin u$, or there is a bisequent $s =: b \Vdash c : d$ from $R^A(\Vdash)$ such that A_s is not true in (u, v). The latter condition amounts to $c \setminus \{A\} \subseteq \overline{u}$, $d \subseteq v$ and $b \subseteq \overline{v}$. Consequently, if $A \in u$, then $: \overline{v} \Vdash \overline{u}, A : v$ by Monotonicity. Therefore, there can be no bimodels (u', v) of $\Vdash$ such that u' is a proper subset of u. Combining this with the earlier condition, we obtain that (u, v) is a positively minimal bimodel of $\Vdash$.

Assume now that there is a positively minimal bimodel (u, v) of $\Vdash$ that is not closed with respect some bisequent s_A. Then $A \in u$ and consequently $: \overline{v} \Vdash \overline{u}, A : v$, since (u, v) is positively minimal. Therefore there must exist a bisequent $s =: b \Vdash c : d$ in $R^A(\Vdash)$ such that $b \subseteq \overline{v}$, $c \setminus \{A\} \subseteq \overline{u}$ and $d \subseteq v$. But then it is easy to check that the formula A_s is not true in (u, v), contrary to the assumption that (u, v) is not closed with respect to s_A. Consequently, all positively minimal bimodels of $\Vdash$ are closed with respect to all the bisequents determining $\Vdash_{K(\Vdash)}$, and hence are bimodels of the latter. Thus, bimodels of $\Vdash_{K(\Vdash)}$ are exactly positively minimal bimodels of $\Vdash$, and therefore $\Vdash_{K(\Vdash)}$ coincides with $\Vdash^c$. $\square$