On the Relationship Between Logic Programming and Non-monotonic Reasoning*

Teodor C. Przymusinski
Department of Mathematics
University of Texas
El Paso, TX 79968
< *ft00@utep.bitnet* >

Abstract

In spite of the existence of a close relationship between logic programming and non-monotonic reasoning, in the past the two research areas have progressed largely independently of each other. Recently, however, a new declarative semantics of logic programs has been proposed and it has been shown to be equivalent to suitable forms of four major non-monotonic formalisms: McCarthy's circumscription, Reiter's closed world assumption, Moore's autoepistemic logic and Reiter's default logic.

The importance of these results stems not only from the fact that they shed new light on the relationship between logic programming and non-monotonic reasoning, but also from the fact that they establish a close relationship between four major formalizations of non-monotonic reasoning for an important class of theories.

1 Introduction

Non-monotonic reasoning and *logic programming* are areas of crucial and growing significance to Artificial Intelligence and to the whole field of computer science. It is therefore important to achieve a better understanding of the relationship existing between these two fields.

Non-monotonic reasoning and logic programming are closely related. The importance of logic programming to the area of non-monotonic reasoning follows from the fact that, as observed by several researchers (e.g. [Reiter 86]), the non-monotonic character of procedural negation used in logic programming often makes it possible to efficiently implement other non-monotonic formalisms in Prolog or in other logic programming languages. Logic programming can also be used to provide formalizations for special forms of non-monotonic reasoning. For example, the calculus of events described in [Kowalski and Sergot 86] uses Prolog's negation as failure operator to formalize the temporal persistence problem in AI.

The importance of the field of non-monotonic reasoning to logic programming is even more apparent. Logic programming is based on the idea of declarative programming stemming from Kowalski's principle of separation of logic and control. Ideally, a programmer should be only concerned with the declarative meaning of his program, while the procedural aspects of the program's execution are handled automatically. Unfortunately, this ideal has not yet been fulfilled. One of the reasons is the lack of clarity as to what should be the proper declarative semantics of logic programs and, in particular, what should be the meaning of negation in logic programming. Logic programs do not use logical negation, but instead rely on a non-monotonic operator – often referred to as *negation as failure* – which represents a procedural form of negation. Without proper declarative semantics the user needs an intimate knowledge of procedural aspects in order to write correct programs. The problem of finding suitable declarative semantics for logic programs can therefore be viewed as the problem of finding a suitable formalization of the type of non-monotonic reasoning used in logic programming.

In spite of this close relationship between non-monotonic reasoning and logic programming, the two research areas are developing largely in parallel rather than in tandem and there is not as much interaction between the two fields as one would expect. One possible explanation of this phenomenon is the fact that, traditionally, the declarative semantics of logic programming has been based on the non-monotonic formalism, developed in [Clark 78], and called Clark's predicate completion (see [Lloyd 84]). Clark's formalism is based on a very intuitive and useful idea of constructing the completion of a program P by essentially replacing the 'if' statements in P by suitable 'iff' statements. Unfortunately, Clark's formalism is not sufficiently general to be applied beyond the realm of logic programming and therefore does not play a major role in formalizing general non-monotonic reasoning in AI. In addition, Clark's approach has some other serious drawbacks often discussed in the literature (see e.g. [Shepherdson 86]).

Recently, however, a new approach to the problem of declarative semantics of logic programs has been proposed and elegant and easily intelligible semantics for such programs has been developed [Apt, Blair and Walker 88; Van Gelder 88; T. Przymusinski 87] It has been shown that the proposed semantics is equivalent to suitable forms of four major non-monotonic formalisms: McCarthy's circumscription, Reiter's closed world assumption, Moore's autoepistemic logic and Reiter's default logic.

The importance of these results is at least twofold. Firstly, they shed new light on the relationship between logic programming and non-monotonic reasoning. Secondly, they establish a close relationship between the four major formalizations of non-monotonic reasoning for an important class of theories. They may also contribute to a better understanding of relations existing between various forms of non-monotonic reasoning and to the eventual discovery of deeper underlying principles of non-monotonic

*The full version of this article will appear in "Handbook on Formal Foundations of AI", D.Partridge and Y.Wilks (editors).

reasoning. The aim of this paper is to present an account of these recent developments.

2 Perfect Model Semantics for Logic Programs

In [Apt, Blair and Walker 88] and [Van Gelder 88] an important class of *stratified logic programs* was introduced, a unique 'natural' minimal Herbrand model M_P of a stratified logic program was defined and it was argued that this model may be taken to represent the declarative semantics of such programs.

In [T. Przymusinski 88] and [T. Przymusinski 87] the class of *perfect models* of a logic program was defined and it was shown that every stratified logic program has exactly one perfect *Herbrand* model which coincides with the model M_P. The *perfect model semantics* of logic programs is the semantics determined by the class PERF(P) of all (not necessarily Herbrand) perfect models of a program P.

We first introduce the *dependency graph* G of the program P whose vertices are predicate symbols occurring in P. If A and B are predicate symbols, then there is a directed edge in G from B to A if and only if there is a clause in P such that A occurs in its head and B in one of its premises. If this premise is negative, then the edge is called *negative*. For any two predicate symbols in P we say that *B has lower priority than A* (briefly, $B < A$) if there is a directed path in G leading from B to A and passing through at least one negative edge. We call the relation defined above *the priority relation* [T. Przymusinski 87].

We now define the notion of a perfect model. It is our goal to *minimize extensions of low priority predicates as much as possible*, and we are willing to do that even at the cost of enlarging extensions of predicates with higher priority. Consequently, if M is a model of P and if a new model N is obtained from M by changing extensions of some predicates in M, then we will consider the new model N to be *preferable* to M if and only if addition of some new element(s) to the extension of a higher priority predicate A is always *justified* by the simultaneous removal of some elements from the extension of a lower priority predicate B, i.e. such that $B < A$. A model M will be considered *perfect* if there are *no* models preferable to it. More formally:

2.1. Definition. [T. Przymusinski 87] Suppose that M and N are two distinct models of a general program P, with the same universe and the same interpretation of functions (and constants) and denote by $E_M(A)$ and $E_N(A)$ the extensions in M and N, respectively, of a predicate A. We say that N is *preferable* to M (briefly, $N \prec M$), if for every predicate A for which the set $E_N(A) - E_M(A)$ is non-empty there is a predicate symbol $B < A$ such that the set $E_M(B) - E_N(B)$ is non-empty. We say that a model M of P is *perfect* if there are no models preferable to M. We call the relation $\prec$ the preference relation between models and we write $M \preceq N$, if $M=N$ or $M \prec N$.

2.2. Theorem. [T. Przymusinski 87] Every perfect model is minimal.

For positive[1] logic programs the converse is true.

2.3. Theorem. [T. Przymusinski 87] If M is a model of a positive logic program then M is minimal if and only if M is perfect.

2.4. Example. Not every logic program has a perfect model. The program $\{p \leftarrow \neg q,\ q \leftarrow \neg p\}$ has only two minimal Herbrand models $M_1=\{p\}$ and $M_2=\{q\}$ and since $p < q$ and $q < p$ we have $M_1 \prec M_2$ and $M_2 \prec M_1$, thus none of these models is perfect.

The cause of this peculiarity is quite clear: our semantics is based on relative priorities between predicate symbols and therefore we have to be consistent when assigning those priorities to avoid *priority conflicts*, which – in the dependency graph G – correspond to cycles passing through negative edges. This leads to the following definition:

2.5. Definition. [Apt, Blair and Walker 88; Van Gelder 88] A logic program P is *stratified* if its dependency graph does not contain cycles passing through negative edges. Equivalently, a logic program P is stratified if and only if it is possible to decompose the set S of all predicates of P into disjoint sets $S_1, \ldots, S_r$, called *strata*, so that for every clause

$$C \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_n$$

in P, where A's, B's and C are atoms, we have that:

(i) for every i, $\text{stratum}(A_i) \leq \text{stratum}(C)$,

(ii) for every j, $\text{stratum}(B_j) < \text{stratum}(C)$,

where stratum(A)=i, if the predicate symbol of A belongs to S_i. Any particular decomposition $\{S_1, \ldots, S_r\}$ of S satisfying the above conditions is called a *stratification of P*.

In the above definition, stratification determines priority levels (strata) of S, corresponding to the priority relation defined before. The following basic result states that every model of a stratified program P is 'subsumed' by a perfect model. This property is exactly analogous to the well-known property of minimal models [Bossu and Siegel 85].

2.6. Theorem. [T. Przymusinski 87] For every model N of a stratified program P there exists a perfect model M such that $M \preceq N$. Moreover, every stratified program has a unique perfect *Herbrand* model which coincides with the model M_P.

Now we define the perfect model semantics of a logic program.

2.7. Definition. [T. Przymusinski 87] Let P be a logic program and let PERF(P) be the set of all perfect models of P. By the *perfect model semantics* of P we mean the semantics induced by the set PERF(P). Under this semantics a sentence F is considered to be true iff F is satisfied in all perfect models of P. In this case we write $PERF(P) \models F$.

Theorem 2.3 implies that for positive logic programs, the perfect model semantics is in fact equivalent to the *minimal model semantics*, i.e. to the semantics induced by the class MIN(P) of all – not necessarily Herbrand – minimal models of P. The *perfect model semantics is stronger than the semantics defined by Clark's completion* comp(P) of the program P, i.e. for any sentence F, if $comp(P) \models F$, then $PERF(P) \models F$. However, as the following example indicates, the perfect model semantics eliminates some of the unintuitive features of Clark's semantics.

[1] A program is positive if it does not have negative premises.

2.8. Example. (Van Gelder). Suppose, that we wanted to describe which vertices in a graph are reachable from a given vertex a. We could write

$$\begin{aligned} &edge(a,b) \quad edge(c,d) \quad edge(d,c) \\ &reachable(a) \\ &reachable(X) \leftarrow reachable(Y), edge(Y,X) \\ &unreachable(X) \leftarrow \neg reachable(X). \end{aligned}$$

This seems to be a very reasonable program and we certainly can expect vertices c and d to be unreachable from a. However, Clark's completion of P lacks the power to derive these conclusions [T. Przymusinski 87]. On the other hand, it is easy to verify that the program is stratified by a stratification S_1={reachable, edge} and S_2={unreachable} and that the perfect model semantics provides the correct answers.

2.1 Procedural semantics: SLS-resolution

In [T. Przymusinski 87] *SLS-resolution (Linear resolution with Selection function for Stratified programs)* was defined and it was shown that SLS-resolution is sound and complete (for non-floundering queries[2]) w.r.t. the perfect model semantics and therefore provides a procedural mechanism for the proposed semantics. SLS-resolution is a natural *generalization* of SLD-resolution (Linear resolution with Selection function for Definite programs) from the class of positive (definite) programs onto the class of stratified programs. SLS-resolution differs from SLDNF-resolution primarily by not relying on finite failure trees.

2.9. Theorem. (Soundness of SLS-resolution) Suppose that P is a stratified program and $G = \leftarrow W$ is a goal. Then

(i) If θ is any SLS-answer-substitution, then $PERF(P) \models W\theta$

(ii) If SLS-tree for G is failed, then $PERF(P) \models \neg W$.

2.10. Theorem. (Completeness of SLS-resolution) Suppose that P is a stratified program, $G = \leftarrow W$ is a non-floundered goal and θ is a substitution. Then

(i) $PERF(P) \models W\theta$ iff there exists an SLS-answer-substitution more general than θ;

(ii) $PERF(P) \models \neg W$ iff SLS-tree for G is failed.

In the special case when P is a positive program, SLS-resolution reduces to the standard SLD-resolution. Theorem 2.10 therefore implies an important result showing that for positive goals *SLD-resolution is sound and complete w.r.t. the minimal model semantics.*

3 Equivalence to Non-monotonic Formalisms

In this section we show that the perfect model semantics for logic programs described in the previous section is (semantically) equivalent to suitable forms of four major non-monotonic formalisms: (1) circumscription, (2) closed world assumption, (3) autoepistemic logic and (4) default logic.

These results provide a further argument in favor of the perfect model semantics and underscore the relationship between logic programming and non-monotonic reasoning. They should also contribute to a better understanding of the relation existing between various forms of non-monotonic reasoning.

3.1 Circumscription

One of the most powerful formalizations of non-monotonic reasoning called circumscription, was introduced in [McCarthy 80; McCarthy 86]. The following theorem establishes the equivalence between the perfect model semantics of logic programs and the semantics of prioritized circumscription. A similar result for pointwise circumscription was obtained in [Lifschitz 88] and related results were obtained earlier in [Reiter 82].

3.1. Theorem. [T. Przymusinski 87] Suppose that P is a stratified program and $S_1,...,S_n$ is a stratification of P. A model of P is perfect if and only if it is a model of prioritized circumscription CIRC(P;$S_1 > ... > S_n$). Consequently, the perfect model semantics of P coincides with the semantics of prioritized circumscription of P, i.e. for any sentence F

$$PERF(P) \models F \Longleftrightarrow CIRC(P; S_1 > ... > S_n) \models F.$$

The above theorem has two interesting consequences:

- Since SLDNF-resolution used in Prolog is sound w.r.t. the perfect model semantics it is also sound w.r.t. to the semantics of prioritized circumscription. This means that SLDNF-resolution can be used as a sound inference engine for some types of circumscription[3]. This formally confirms Reiter's comment that 'partly because it is a non-monotonic operator, procedural negation can often be used to implement other forms of non-monotonic reasoning' [Reiter 86].
- In general, it is not clear how to instantiate the circumscription axiom in order to derive the desired consequences of a circumscribed theory. The equivalence between the perfect model semantics and prioritized circumscription shows that in the case of stratified logic programs such an instantiation can be generated automatically based on the syntactic form of the program.

3.2 Closed world assumption

An alternative way to formalize non-monotonic reasoning is to use some form of the closed world assumption. The first step in this direction was made by Reiter, who defined the so called *naive closure* CWA(P) of a theory P:

3.3. Definition. [Reiter 78] The *naive closure* CWA(P) of P is defined as follows:

$$CWA(P) = P \cup \{\neg p: \text{ p is a ground atom and } P \not\models p\}.$$

Reiter's CWA(P), although suitable for positive programs, is usually inconsistent for programs with negative

[2] See [Lloyd 84].

[3] A query answering algorithm for general circumscriptive theories has been described in [T. Przymusinski 87a].

premises. For example, if P is $p \leftarrow \neg q$, then CWA(P) implies $\neg p$ and $\neg q$ and is inconsistent.

Stimulated by Reiter's work, several researchers proposed more sophisticated forms of the closed world assumption, namely the Generalized Closed World Assumption [Minker 82], the Extended Closed World Assumption [Gelfond, H. Przymusinska and T. Przymusinski 86a; Yahya and Henschen 85] and – the most general of them all – the Iterated Closed World Assumption (ICWA) [Gelfond, H. Przymusinska and T. Przymusinski 86a]. Below, we give a simplified definition of ICWA, which is adequate for logic programs.

Let P be a stratified logic program with a stratification $\{S_1, \ldots, S_k\}$. For every n, let $Q_n = \bigcup\{S_j : j \leq n\}$ and let P_n be the logic program consisting of all clauses from P whose heads involve predicates from Q_n. Clearly, $P=P_k$.

3.4. Definition. [Gelfond, H. Przymusinska and T. Przymusinski 86a] The Iterated Closed World Assumption applied to P is defined as the closure ICWA(P; $S_1 > \ldots > S_k$) of P obtained by iteration of the appropriate CWA's:

$$ICWA(P_1; S_1) = CWA(P_1);$$

$$ICWA(P_{n+1}; S_1 > \ldots > S_{n+1}) =$$

$$= CWA(P_{n+1} + ICWA(P_n; S_1 > \ldots > S_n)), \text{ for } n > 0,$$

$$ICWA(P; S_1 > \ldots > S_k) = ICWA(P_k; S_1 > \ldots > S_k).$$

The following theorem shows that the semantics of ICWA(P;$S_1 > \ldots > S_k$) is equivalent to the perfect model semantics of P:

3.5. Theorem [Gelfond, H. Przymusinska and T. Przymusinski 86a] Assume the domain closure axiom and suppose that P is a stratified logic program and $S_1,\ldots,S_k$ is a stratification of P. The theory ICWA(P;$S_1 > \ldots > S_k$) has exactly one model and this model is the unique perfect model of P.

The above theorem provides a syntactic description of the perfect model semantics in the form of a first order theory. It generalizes an earlier result obtained for positive programs and minimal models [Lifschitz 85].

3.3 Autoepistemic logic

Autoepistemic logic (AEL) proposed in [Moore 80] provides another interesting formalization of non-monotonic reasoning. Moore uses modal logic to formalize an agent's reasoning about his knowledge or beliefs. We will follow Moore and consider here propositional theories only.

By an autoepistemic theory T we mean a set of formulae in the language of propositional calculus augmented by a belief operator L, where, for any formula F, LF is interpreted as 'F is believed'. The set of all propositional consequences of T will be denoted by Th(T).

The central role in Moore's formalization is played by the notion of a stable autoepistemic expansion of T which intuitively represents a possible set of beliefs of an ideally rational agent. The agent is ideally rational in the sense that he believes in all and only those facts which he can conclude from T and from his other beliefs. If this expansion is unique then it can be viewed as the set of theorems which follow from T in the autoepistemic logic.

3.6. Definition. [Moore 80] A set of formulae E(T) is a *stable autoepistemic expansion* of T if it satisfies the following fixed point condition:

$$E(T) = Th(T \cup \{Lp : p \in E(T)\} \cup \{\neg Lp : p \notin E(T)\}),$$

where p is a propositional formula.

To establish a relationship between perfect model semantics and autoepistemic logic we have to define an interpretation of propositional formulae in terms of autoepistemic beliefs.

3.7. Definition. [Gelfond 78] For any propositional formula F the interpretation I(F) of F is obtained by replacing every occurrence of a negative literal $\neg p$ in F by the negative autoepistemic literal $\neg Lp$. For a logic program P, by I(P) we denote the set of all autoepistemic formulae I(F), where F is a clause from P.

The following theorem shows that – under the above interpretation – autoepistemic logic is semantically equivalent to the perfect model semantics. It has been shown that this equivalence holds precisely for the class of stratified logic programs [H. Przymusinska 87].

3.8. Theorem. [Gelfond 78] If P is a stratified logic program, then autoepistemic logic and perfect model semantics are I-equivalent. More precisely, the theory I(P) has a unique stable autoepistemic expansion E(I(P)) and for every formula F, we have: $PERF(P) \models F$ iff $E(I(P)) \models I(F)$.

The above theorem underscores an important feature of autoepistemic logic, namely the fact that in order to obtain equivalence with the perfect model semantics, it is not necessary to introduce the concept of prioritization into autoepistemic logic as it was the case with circumscription and the closed world assumption. In a sense, prioritization is already built into autoepistemic logic.

As it was the case for circumscription, the above theorem implies that SLDNF-resolution can be used as a sound deductive mechanism for a class of autoepistemic theories. This can be useful in view of the fact that autoepistemic logic was presented non-constructively and no procedural mechanism was provided to derive its theorems.

3.4 Default logic

Another approach to the formalization of non-monotonic reasoning was proposed in [Reiter 80] and called default logic. Its distinguishing feature is the introduction of default statements which function as additional rules of inference rather than formulae in some theory.

A (closed) *default rule* R is a rule of the form

$$\frac{a : Mb_1, \ldots, Mb_n}{c}$$

where a, b_i's and c are closed first order formulae and the intuitive meaning of R is that 'if a holds and if each one of b_i's can be consistently assumed, then c can be inferred'.

A *default theory* is a pair $< D, T >$, where D is a set of default rules and T is a set of closed first order formulae. T describes facts which are known to be true and the default rules enable us to derive from them additional information which, together with T itself, constitutes a more complete extension of the theory. The extensions are defined using the fixed point construction:

3.9. Definition [Reiter 80] Suppose that $< D, T >$ is a default theory. For any set S of first order sentences define G(S) to be the smallest set (it always exists!) with the following properties:

(i) T is contained in G(S);

(ii) G(S) is closed under logical consequence;

(iii) if R is a rule from D (of the above described form) and if a is in G(S) and, for every i, $\neg b_i$ is not in S, then c is in G(S).

Any set of first order sentences E satisfying E=G(E) is called an *extension* of $< D, T >$, i.e. extensions are fixed points of the operator G.

A default theory $< D, T >$ may have none, one, or more than one extension. Any such extension is a possible set of beliefs for an agent. If the theory has exactly one extension E, then E can be considered as the set of theorems of $< D, T >$.

In [Bidoit and Froidevaux 86] it was shown that the perfect model semantics of a stratified logic program P is equivalent to a suitable default theory generated by P. Suppose that P is a logic program. Denote by T the set of all positive clauses of P and by D the set of defaults obtained as follows: for any clause

$$C \leftarrow A_1, ..., A_m, \neg B_1, ..., \neg B_n,$$

in P such that $n > 0$, include in D the default rule:

$$\frac{A_1 \wedge ... \wedge A_m : M\neg B_1, ..., M\neg B_n}{C}$$

and call the resulting default theory $< D, T >$ the *default theory associated with the program P.*

3.10. Theorem. [Bidoit and Froidevaux 86] Suppose that P is a stratified logic program and $< D, T >$ is the associated default theory. The theory $< D, T >$ has exactly one extension E and the unique minimal model of E is the unique perfect model of P.

The above approach is similar to that used in the case of autoepistemic logic, which is not surprising in view of the close relationship existing between default and autoepistemic logics [Konolidge 87].

References

[Apt, Blair and Walker 88] Apt, K., Blair, H. and Walker, A., 'Towards a Theory of Declarative Knowledge', in: Foundations of Deductive Databases and Logic Programming, (ed. J.Minker), Morgan Kaufmann 1988, 89-148.

[Bidoit and Froidevaux 86] Bidoit, N. and Froidevaux, C., 'Minimalism Subsumes Default Logic and Circumscription in Stratified Logic Programming', preprint, 1986.

[Bossu and Siegel 85] Bossu, G. and Siegel, P., 'Saturation, Nonmonotonic Reasoning and the Closed World Assumption', Artificial Intelligence 25(1985), 13-63.

[Clark 78] Clark, K.L., 'Negation as Failure', in: Logic and Data Bases (H.Gallaire and J.Minker, Eds.), Plenum Press, New York 1978, 293-322.

[Gelfond 78] Gelfond, M., On Stratified Autoepistemic Theories, Proceedings AAAI-87.

[Gelfond and H. Przymusinska 86] Gelfond, M. and Przymusinska, H., 'Negation as Failure: Careful Closure Procedure', Artificial Intelligence 30(1986), 273-287.

[Gelfond, H. Przymusinska and T. Przymusinski 86a] Gelfond, M., Przymusinska, H. and Przymusinski, T., 'On the Relationship between Circumscription and Negation as Failure', Artificial Intelligence, to appear.

[Konolidge 87] Konolige, K., 'On the relation between default theories and autoepistemic logic', SRI International, 1987, draft paper.

[Kowalski and Sergot 86] Kowalski, R. and Sergot, M., 'A Logic-based Calculus of Events', New Generation Computing 4(1986), 67-95.

[Lifschitz 85] Lifschitz, V., 'Closed World Data Bases and Circumscription', Artificial Intelligence 27(1985), 229-235.

[Lifschitz 88] Lifschitz, V., 'On the Declarative Semantics of Logic Programs with Negation', in: Foundations of Deductive Databases and Logic Programming, (ed. J.Minker), Morgan Kaufmann 1988, 177-192.

[Lloyd 84] Lloyd, J.W., Foundations of Logic Programming, Springer-Verlag 1984.

[McCarthy 80] McCarthy, J., 'Circumscription - a Form of Non-Monotonic Reasoning', Artificial Intelligence 13(1980), 27-39.

[McCarthy 86] McCarthy, J., 'Applications of Circumscription to Formalizing Common Sense Knowledge', J. Artificial Intelligence 28(1986), 89-116.

[Minker 82] Minker, J., 'On Indefinite Data Bases and the Closed World Assumption', Proc. 6-th Conference on Automated Deduction, Springer Verlag, 1982, 292-308.

[Moore 80] Moore, R.C., 'Semantic Considerations on Nonmonotonic Logic', Artificial Intelligence 25(1985), 75-94.

[H. Przymusinska 87] Przymusinska, H., ' On the Relationship between Autoepistemic Logic and Circumscription for Stratified Deductive Databases', Proceedings of the International Symposium on Methodologies for Intelligent Systems, Knoxville 1987.

[T. Przymusinski 87] Przymusinski, T., 'On the Declarative and Procedural Semantics of Logic Programs', to appear.

[T. Przymusinski 87a] Przymusinski, T., 'An Algorithm to Compute Circumscription', Artificial Intelligence, to appear.

[T. Przymusinski 88] Przymusinski, T., 'On the Declarative Semantics of Stratified Deductive Databases and Logic Programs', in: Foundations of Deductive Databases and Logic Programming (ed. J.Minker), Morgan Kaufmann 1988, 193-216.

[Reiter 78] Reiter, R., 'On Closed-World Data Bases', in: Logic and Data Bases (H.Gallaire and J.Minker, Eds.), Plenum Press, New York 1978, 55-76.

[Reiter 80] Reiter, R., 'A Logic for Default Theory', Artificial Intelligence 13(1980), 81-132.

[Reiter 82] Reiter, R., 'Circumscription implies Predicate Completion (sometimes)', Proc. AAAI-82, 1982, 418-420.

[Reiter 86] Reiter, R., 'Nonmonotonic Reasoning', Annual Reviews of Computer Science, to appear.

[Shepherdson 86] Shepherdson, J., 'Negation in Logic Programming', J. Logic Programming, to appear.

[Van Gelder 88] Van Gelder, A., 'Negation as Failure Using Tight Derivations for General Logic Programs', in: Foundations of Deductive Databases and Logic Programming, (ed. J.Minker), Morgan Kaufmann 1988, 149-176.

[Yahya and Henschen 85] Yahya, A. and Henschen, L., 'Deduction in Non-Horn Databases', Journal of Automated Reasoning 1(2)(1985),141-160.