Disjunctive Logic Programming with Negation As Failure in the Head

Laura A. Cecchi[*]

Departamento de Informática y Estadística
Facultad de Economía y Administración
Universidad Nacional del Comahue
`e-mail:lcecchi@uncoma.edu.ar`

Pablo R. Fillottrani Guillermo R. Simari

Grupo de Investigación en Inteligencia Artificial
Departamento de Ciencias de la Computación
Universidad Nacional del Sur
`e-mail: {ccfillo, grs}@criba.edu.ar`

Keywords: Disjunctive logic programming - Negation as failure in head - Non minimal model semantics - Answer set semantic

Abstract

Disjunctive logic programs have been studied in order to increase expressivity, especially in representing indefinite information. Even though there is no general consensus on any specific semantic for disjunctive logic programs with negation, most of well known semantics are minimal model based. Sometimes real applications need to capture non minimal models, in spite of the fact that they contain redundant information. In representing disjunctive knowledge, for instance, minimal approaches capture exclusive disjunction but they cannot managed the inclusive meaning.

The purpose of this paper is to analyze the expressive power of a special combination of disjunctive logic programming, negation as failure (NAF), and strong negation. This class of programs are syntactically uniform, since both kinds of negation are allowed to appear in the head and in the body of a rule, and its semantics allows non minimal models. We introduce a non injective syntactic transformation of disjunctive logic programs with NAF in head into disjunctive logic programs without NAF in head, in order to compare semantically both class of programs.

Finally, we prove that the set of answer sets of the transformed disjunctive logic program is strictly included in the set of minimal answer sets of the original program and we discuss its consequences.

[*]Supported by a fellowship of the Universidad Nacional del Comahue.

Disjunctive Logic Programming with Negation As Failure in the Head

1 Introduction

Logic programs are nowadays widely recognized as a valuable tool for knowledge representation and commonsense reasoning. In order to increase the applicability of logic programming in many fields, several extensions of the class of definite programs have been proposed. Normal programs [Llo87] add negation as failure in program clause bodies, basic programs [GL90] allow strong negation not only in the clause bodies but in the clause heads as well and disjunctive programs [Min82] allow disjunction as heads of program clauses.

Disjunctive logic programs have been studied in order to increase expressivity, especially in representing indefinite information. Even though there is no general consensus on any specific semantic for disjunctive logic programs with negation, most of well known semantics are minimal model based. Several model classes defined, such as the perfect and stable models for theories with body negation are subsets of the minimal models and coincide with the minimal model in the absence of negation in the rule bodies. Sometimes real applications need to capture non minimal models, in spite of the fact that they contain redundant information. In representing disjunctive knowledge, for instance, minimal approaches capture exclusive disjunction but they cannot managed the inclusive meaning.

The purpose of this paper is to analyze the expressive power of a special combination of disjunctive logic programming, negation as failure (NAF), and strong negation. This class of programs are syntactically uniform, since both kinds of negation are allowed to appear in the head and in the body of a rule, and its semantics allows non minimal models. This idea was introduced by Liftchitz and Woo [LW92]. Arguing the lack of uniformity of logic programming, Liftchitz [Lif94] defined the *logic of minimal belief and negation as failure* which provides a unified framework for several logic programming languages and non monotonic formalisms.

In [LW92], a class of propositional formulas of minimal belief and negation as failure called theories with protected literals (PL-Theories), is defined. Protected literals are formulas of the form $\mathbf{B}L$ and $not\ L$, where L is a literal and, $\mathbf{B}$ and not are two non monotonic operators representing minimal belief and NAF respectively. A PL-theory is a set of PL-formula which can be characterized by formulas that can be built from protected literals using $\neg$, $\mathbf{B}$, not and $\wedge$. The class of disjunctive programs with NAF in the head was introduced as an equivalent way of writting a disjuntion of protected literals and their negation. Therefore disjunctive logic programs with NAF in head are embedded into minimal belief and negation as failure.

In this paper we compare disjunctive programs with NAF in the head with disjunctive logic programs without NAF in head. In section 2, we review some background about this class of logic programs. Section 3 describes and illustrates the answer set semantics, discusses its non minimality property and shows how it can capture the inclusive meaning. Section 4 introduces a non injective syntactic transformation of disjunctive logic programs with NAF in head into disjunctive logic programs without NAF in head and compares semantically both class of programs, specifying those programs with NAF in head which have no equivalent translations without NAF in head. Then in section 5, we present the main result of this paper. We prove that the set of answer sets of the transformed disjunctive logic program is strictly included in the set of minimal answer sets of the original program and we discuss its consequences. Finally in section 6, we give our conclusions and mention some possible directions for future research.

2 Programs with NAF in the Head

In [Wag94] Gerd Wagner distinguishes three different notions of negation:

- *Default Implicit Falsity* or *Weak Negation*: Defined through the *negation as failure (NAF)* operator which is represented by the symbol *not*. NAF is used to represent incomplete information.
- *Directly Established Falsity* or *Strong Negation*[1]: This notion of negation is represented by the symbol $\neg$. The strong negation of an atom holds if the atom is explicitly false.
- *Negation as Inconsistency*: An atom is false if its addition to the knowledge base leads to inconsistency.

The class of programs we will study deals only with *weak* and *strong* negation. Next we present the syntax of these programs.

Definition 2.1. A *disjunctive program with negation as failure* is a set of rules of the form

$$L_1| \ldots |L_k|\ not\ L_{k+1}|\ \ldots\ |not\ L_m \leftarrow L_{m+1} \wedge \ \ldots \wedge L_r \wedge\ not\ L_{r+1} \wedge\ \ldots \wedge\ not\ L_n \tag{1}$$

where the symbol $|$ stands for $\vee$, L_i's is either a ground atom A or its strong negation $\neg A$ and $n \geq r \geq m \geq k \geq 0$. The left-hand side of the rule is called *Head*, while the right-hand side of the rule is called *Body*. The set of all disjunctive programs with NAF will be denote by $\mathbf{DP}^{NAF}$. Those programs such that $k = m$ and $r = n$ will be called *positive disjunctive programs*. The set of all positive disjunctive programs will be denoted by **DP**. Finally, we will denote the set of all disjunctive programs with $k = m$ (*i.e.*, with negation as failure only in the body of the rules) by **GDP**. ■

A formula of the form (1) with $m = k = 0$ is called *constraint*. A set X of literals *violates* a constraint if $\{L_{m+1}, \ldots, L_r\} \subseteq X$ and $X \cap \{L_{r+1}, \ldots, L_n\} = \varnothing$. Otherwise, X satisfies the constraint. Constraints will not be considered in particular, even though all definitions and results include them.

We use $|$ rather than $\vee$ in disjunctive rule heads because there is a subtle difference between the use of disjunction in the heads of rules and the use of disjunction in classical logic. Consider the following program from [GL91] $\{q \leftarrow p, p|\neg p \leftarrow\}$. The disjunctive rule expresses that p is either known to be true or known to be false. Intuitively, every model of this program includes either p or $\neg p$ and therefore the possible minimal models

[1]In some of the literature strong negation is called *classical negation*. Answer set semantic is not "contrapositive" with respect to $\leftarrow$ and $\neg$, *i.e.*, it assigns different meanings to the rules $p \leftarrow \neg q$ and $q \leftarrow \neg p$. As interaction between strong negation and $\leftarrow$ is different from the interaction between classical negation and $\leftarrow$, we prefer *strong negation* rather than *classical negation* in order to keep in mind this distinction.

are $\{p, q\}$ and $\{\neg p\}$. Unlike the *law of the excluded middle* $(A \vee \neg A)$ in classical logic, the rule $p|\neg p$ cannot be removed from the program without changing its meaning.

Let's present some notation. For any set X of literals, we will denote the set $\{not\ L : L \in X\}$ by $not(X)$. Henceforth, a disjunctive rule will be represented in the form [Lif96]

$$HPos \cup not(HNeg) \leftarrow BPos \cup not(BNeg) \tag{2}$$

where $HPos$, $HNeg$, $BPos$, $BNeg$ are some finite sets of literals. In particular, (1) is represented by setting

$$\begin{aligned} HPos &= \{L_1, \ldots, L_k\} \\ HNeg &= \{L_{k+1}, \ldots, L_m\} \\ BPos &= \{L_{m+1}, \ldots, L_r\} \\ BNeg &= \{L_{r+1}, \ldots, L_n\} \end{aligned}$$

By definition a program is a set of atomic rules. However, real application programs usually use variables in order to express general information. Following Liftchitz's notation [Lif96], literals that contain variables will be called schematic literals. Schematic rule and schematic program definitions are parallel to the definitions of a rule and of a program given in definition 2.1, with schematic literals instead of literals.

Let R be a schematic rule and P a schematic program. $Ground(R)$ stands for the set of all ground instances of R and

$$Ground(P) = \bigcup_{R \in P} Ground(R)$$

Henceforth, we will use the terms program and $Ground(P)$ which clearly is a program in the sense of the definition 2.1 interchangeably.

Example 2.2. We would like to represent the fact that a person who likes literature either reads Bécquer's work or Bradbury's work. Furthermore, we want to distinguish between people who like fiction and romantic books. The following schematic program encodes the above intended meaning, together with the fact that Jean is a person who likes literature. Note that $Ground(P) \in \mathbf{DP}$.

$reads(X, becquer) \mid reads(X, bradbury) \leftarrow person(X), likes\_literature(X)$
$likes(X, fiction) \leftarrow reads(X, bradbury)$
$likes(X, romantic) \leftarrow reads(X, becquer)$
$person(jean) \leftarrow$
$likes\_literature(jean) \leftarrow$

■

Example 2.3. Suppose we want to encode the operation of switch that turns on a light. Assuming that the light and the switch are not prone to be broken, we want to represent that the light will be on if the switch is on, and that the switch will be on whenever the light is on. The following schematic program P represents this information and the fact that a person will fall down if the light is not on. $Ground(P)$ is in$\mathbf{DP}^{NAF}$.

$$\begin{aligned} P: \quad & light\_on \mid not\ switch\_on \leftarrow \\ & switch\_on \mid not\ light\_on \leftarrow \\ & falls\_down(X) \leftarrow person(X) \wedge not\ light\_on \\ & person(marie) \leftarrow \end{aligned}$$

■

Next section formalizes this intuitive meaning of disjunctive programs through the answer set semantic.

3 Disjunctive Answer Set

In this section we introduce the semantics of the programs in **GDP**. The declarative meaning of logic programming has been studied based on the sceptical semantics. In [GL88], Gelfond and Lifschitz defined the stable semantic based on the minimal model semantic which is first extended to programs with strong negation [GL90], and then to **DP**[GL91].

Definition 3.1. [Lif96] Let $P \in \mathbf{DP}$ and X be a set of literals. X is said to be *logically closed* if it is consistent or X contains a pair of complementary literals[2] and therefore X is equals to the set of all grounded literals, that we will denote Lit. X of literals is said to be *closed under* P if for every disjunctive rule $Head \leftarrow Body$ in P, $Head \cap X \neq \varnothing$ whenever $Body \subseteq X$. X is an *answer set* for P if it is a minimal (relative to set inclusion) set of literals that is both closed under P and logically closed. ■

Example 3.2. Let's consider again the program in example 2.2. The possible answer set of P are

$$\{person(jean), reads(jean, becquer), likes(jean, romantic)\}$$

and

$$\{person(jean), reads(jean, bradbury), likes(jean, fiction)\}$$

As the property of minimality holds under the answer set semantic, a person cannot read both Bécquer and Bradbury, which is our intended meaning. ■

Definition 3.3. [Lif96]Let P be in $\mathbf{DP}^{NAF}$. The *reduct* of P relative to a set of literals X is obtained from X by:

- deleting each disjunctive rule (2) such that $HNeg \not\subseteq X$ or $BNeg \cap X \neq \varnothing$, and
- replacing each remaining disjunctive rule (2) by $HPos \leftarrow BPos$.

This program in **DP** will be denoted by P^X.
X is an *answer set* for P if X is an answer set for P^X. A consequence of P is a literal that belongs to all its answer sets. ■

Example 3.4. Let's consider the example 2.3. The answer sets of P are:

$$X_1 = \{person(marie), falls\_down(marie)\}$$

and

$$X_2 = \{person(marie), light\_on, switch\_on\}$$

where the reducts are

[2]Let A be an atom. The literals A and $\neg A$ are said to be complementary.

$$\begin{array}{ll} P^{X_1}\colon & falls\_down(marie) \leftarrow person(marie) \\ & person(marie) \leftarrow \end{array}$$

and

$$\begin{array}{ll} P^{X_2}\colon & light\_on \leftarrow \\ & switch\_on \leftarrow \\ & person(marie) \leftarrow \end{array}$$

■

In view of example 3.4, we can informally describe the meaning of the *not* in the head of a rule as follows. If the set of consequences of a program does not include $HNeg$, then it will exist $H \in HNeg$ such that it is not a consequence of the program. Therefore, $notH$ and the disjunction will hold. So we do not need to take into account this rule. In the example 2.3 under the first answer set the disjunction $light\_on \mid not\ switch\_on$ holds because $not\ switch\_on$ holds. If $HNeg$ is included in the set of consequences of a program, then the truth value of the disjunction will depend only on $HPos$. The second answer set of the program P introduced in the example 2.3 reflects this situation.

Lifschitz and Woo pointed out that minimality does not hold in the answer set semantic of a program in $\mathbf{DP}^{NAF}$. For instance, let's consider the following program P with only one rule $\{r \mid not\ r\}$. If $X = \{r\}$ then $P^{\{r\}} = \{r\}$ and if $X = \{\}$ then $P^{\{\}} = \{\}$. Both $\{r\}$ and $\{\}$ are answer sets.

The non minimality property is useful to distinguish between the inclusive and exclusive meaning of disjunction. Suppose we want to represent the following disjunction $a \mid b$ whose intended meaning is inclusive. We can translate $a \mid b$ into

$$\begin{array}{ll} a \mid not\ a & \leftarrow \\ b \mid not\ b & \leftarrow \\ & \leftarrow not\ a \wedge\ not\ b \end{array}$$

The answer sets are $\{a\}$, $\{b\}$ and $\{a,\ b\}$, capturing the intended inclusive meaning.

Possible model semantics [Sak89, Cha89] and the mixed completion theory [DL94] are alternatives approaches proposed to solve this problem for positive disjunctive programs. Hence the importance of this application is that it is possible to make an explicit distinction between inclusive and exclusive meaning, in a more general class of programs.

4 Transformation

Extending logic programming with NAF in the head increases the expressivity allowing non minimal answer sets. Therefore given a program in $\mathbf{DP}^{NAF}$, we cannot find an equivalent program in **GDP**. However, we are interested in the semantic relationship between them. In order to investigate this problem, we now define a syntactic transformation and show some examples of its application and consequences.

Definition 4.1. Let $P \in \mathbf{DP}^{NAF}$. The transformation Φ:$\mathbf{DP}^{NAF} \rightarrow$ **GDP**, is defined as

$$\begin{array}{lll} \Phi(P) = \{ & HPos \leftarrow BPos \cup HNeg \cup not\ BNeg \mid \\ & HPos \cup not\ HNeg \leftarrow BPos \cup not\ BNeg \text{ is in } P\} \end{array}$$

■

Informally the transformation Φ considers $HNeg$ as a conditional part of the rule, so that $HNeg$ is removed from the head and is added to the body. The following example will help us to discuss the consequences of applying Φ over a program in $\mathbf{DP}^{NAF}$.

Example 4.2. Let P be the program introduced in the example 2.3. Then $\Phi(P)$ is

$$\begin{array}{ll} \Phi(P): & light\_on \leftarrow switch\_on \\ & switch\_on \leftarrow light\_on \\ & falls\_down(marie) \leftarrow person(marie) \wedge \mathit{not}\ light\_on \\ & person(marie) \leftarrow \end{array}$$

The semantic of $\Phi(P)$ has been restricted to the unique answer set

$$\{person(marie), falls\_down(marie)\}$$

■

The following example shows that the previously defined transformation is not injective.

Example 4.3. $\Phi(P')$ is equal to $\Phi(P)$, where P is the program of the example 2.3.

$$\begin{array}{ll} P': & light\_on \leftarrow switch\_on \\ & switch\_on \leftarrow light\_on \\ & falls\_down(marie) \mid \mathit{not}\ person(marie) \leftarrow \mathit{not}\ light\_on \\ & person(marie) \leftarrow \end{array}$$

Note that in this case the unique answer set of P' is the answer set of $\Phi(P')$. ■

In view of the above examples, we can conclude that there exist some programs in $\mathbf{DP}^{NAF}$ that are equivalent to their transformed programs. At this point, the problem of specifying this subset arises. Programs in $\mathbf{DP}^{NAF}$ that capture a non minimal answer set do not have an equivalent program in **GDP**. For instance, $\{p \mid \mathit{not}\ p\}$ which can be read "p is believed or p is not believed", has two answer sets $\{p\}$ and $\{\}$, but $\{p \leftarrow p\}$ which can be read "p is believed if p is believed", has only one answer set $\{\}$.

Another example is $\{p \mid \mathit{not}\ q, q \mid \mathit{not}\ p\}$. $\{p, q\}$ and $\{\}$ are its answer sets. However, $\Phi(\{p \mid \mathit{not}\ q, q \mid \mathit{not}\ p\}) = \{p \leftarrow q, q \leftarrow p\}$ has an unique answer set, $\{\}$.

These examples have a feature in common: "not" in the head is part of an infinite loop. These programs in $\mathbf{DP}^{NAF}$ have at least two answer sets. The non minimal one includes all the literals in the loop; the minimal one includes none of them.

A transformed program in **GDP** captures only minimal answer set. In the following section we will show that it cannot even capture all minimal answer sets of the original program in $\mathbf{DP}^{NAF}$.

5 Comparing $\mathbf{DP}^{NAF}$ with GDP

In light of the discussion of the previous section, we can assure that including NAF in head increases the expressive power. Since programs in **GDP** capture only minimal models, we can ask ourself if the set of transformed programs are powerful enough to capture all minimal answer sets of the original program in $\mathbf{DP}^{NAF}$. In this section, we will prove that the set of answer sets of transformed programs is strictly included in the set of minimal answer sets of the original programs in $\mathbf{DP}^{NAF}$.

The following lemma is necessary to prove theorem 5.2 which is the main result of the paper.

Lemma 5.1. Let P be a $\mathbf{DP}^{NAF}$. S' is closed under $P^{S''}$ for some S'' such that $S' \subseteq S'' \subseteq S$, then S' is closed under $\Phi(P)^S$. ■

Proof: Let

$$HPos \leftarrow BPos \cup HNeg \ \in \ \Phi(P)^S \tag{3}$$

then exists

$$HPos \cup \ not \ HNeg \leftarrow BPos \cup \ not \ BNeg \ \in \ P$$

such that $BNeg \cap S = \varnothing$.
Let's consider the following cases:

(a) If $BPos \cup HNeg \subseteq S'$ then $BPos \cup HNeg \subseteq S''$. Therefore,

$$HPos \leftarrow BPos \ \in P^{S''}$$

Since S' is closed under $P^{S''}$ and $BPos \subset S', HPos \cap S' \neq \varnothing$ then (3) is satisfied by S'.

(b) If $BPos \cup HNeg \not\subseteq S'$ then S' satisfies (3).

$\therefore$ S' is closed under $\Phi(P)^S$. ■

Theorem 5.2. Let P be a $\mathbf{DP}^{NAF}$. If S is a answer set of $\Phi(P)$ then S is a minimal answer set of P.

Proof: Let S be an answer set of $\Phi(P)$. In order to prove that S is an answer set of P, we must prove that S is an answer set of P^S, *i.e.*,

(i) S is logically closed. Trivial.

(ii) S is closed under P^S.
Let $HPos \leftarrow BPos \ \in \ P^S$, such that $BPos \subseteq S$. Then exists

$$HPos \cup not \ HNeg \leftarrow \ BPos \cup not \ BNeg \ \in \ P$$

such that $HNeg \subseteq S$ and $BNeg \cap S = \varnothing$. So

$$HPos \leftarrow \ BPos \cup HNeg \ \cup \ not \ BNeg \ \in \ \Phi(P)$$

and

$$HPos \leftarrow BPos \ \cup \ HNeg \in \ \Phi(P)^S$$

As S is an answer set of $\Phi(P)$, is an answer set of $\Phi(P)^S$ and S is closed under $\Phi(P)^S$. Therefore $HPos \cap S \neq \varnothing$.

(iii) There is no $S' \subset S$ such that S' satisfies (i) and (ii).
Let's suppose that exists S'. As S' satisfies (ii) then by lemma 5.1 S' is closed under $\Phi(P)^S$. Therefore S' satisfies (i) and is closed under $\Phi(P)^S$ what contradicts the fact that S is an answer set of $\Phi(P)$.

It remains to prove that S is a minimal answer set of P.
Let S' be an answer set of P such that $S' \subset S$. If S' is an answer set of P then S' is logically closed and S' is closed under $P^{S'}$. By lemma 5.1 S' is closed under $\Phi(P)^S$ and this contradicts the fact that S is an answer set of $\Phi(P)$.
$\therefore$ S is a minimal answer set of P. ■

We proved that semantic of the class of transformed programs in **GDP** is always captured by the original program in $\mathbf{DP}^{NAF}$. Furthermore, we proved that an answer set of the transformed program is a minimal answer set of the original program in $\mathbf{DP}^{NAF}$. The converse of this theorem does not hold as the following example shows.

Example 5.3. Consider the following program P and its transformed $\Phi(P)$

$$\begin{array}{llll} P: & q \mid not\ p & \leftarrow & \\ & p \mid not\ q & \leftarrow & \\ & r & \leftarrow not\ p & \end{array} \qquad \begin{array}{ll} \Phi(P): & q \leftarrow p \\ & p \leftarrow q \\ & r \leftarrow not\ p \end{array}$$

$\{p,q\}$ and $\{r\}$ are both minimal answer sets of P. However only $\{r\}$ is answer set of $\Phi(P)$. Once more, NAF in head involved in a loop causes this behavior. ■

The above results show the key distinction between encoding information by $p \mid not\ q$ and encoding it by $p \leftarrow\ q$. The first rule allows us to represent non minimal models and even some minimal models that the second one is not able to express. Its uses will strongly depend on whether the application accepts redundant information or not.

6 Conclusion and Future Work

We have semantically compared programs in $\mathbf{DP}^{NAF}$ and programs in **GDP**, defining a syntactic transformation between them. We have showed that negation as failure in the head increases the expressive power of disjunctive logic programming, since it can capture non minimal answer sets. Furthermore, we have proved that all transformed program answer sets are strictly included in the set of minimal answer sets of the program in $\mathbf{DP}^{NAF}$ been transformed. Hence, transformed programs are not powerful enough to capture all the minimal answer sets of the original program in $\mathbf{DP}^{NAF}$.

We have presented as an immediately application the distinction between the inclusive and exclusive meaning. Representing the inclusive meaning requires to capture redundant information, which is done by the non minimal answer sets.

Recently, Inoue and Sakama [IS98] have independently studied the class of disjunctive programs with NAF in head, restricting the subset of $\mathbf{DP}^{NAF}$ which are semantically equivalent to the class of programs in **GDP**. Furthermore, they have showed that $\mathbf{DP}^{NAF}$ computational complexity remain in the same complexity class as normal disjunctive programs.

Dix in [Dix95a, Dix95b, Dix95c] presents a set of properties for classifying and characterizing the various semantics of logic programs with negation. A possible future work would be to decide whether it is possible to provide a unique characterization of the answer set semantic of disjunctive logic programs with NAF in head.

References

[Cha89] E. P. F. Chan. A Possible World Semantics for Disjunctive Databases. Technical Report CS-89-47, Dept. of Computer Science, University of Waterloo, 1989.

[Dix95a] Jürgen Dix. A classification theory of semantics of normal logic programs: I. Strong properties. *Fundamenta Informaticae*, XXII(3):227–255, 1995.

[Dix95b] Jürgen Dix. A classification theory of semantics of normal logic programs: II. Weak properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.

[Dix95c] Jürgen Dix. Semantics of logic programs: Their intuitions and formal properties. An overview. In André Fuhrmann and Hans Rott, editors, *Logic, Action and Information*, pages 227–313. Gruyter, Berlin-New York, 1995.

[DL94] Phan Minh Dung and Ngo Huu Liem. Negation as failure for disjunctive logic programming. *Annals of Mathematics and Artificial Intelligence*, 12:25–51, 1994.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

[GL90] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In David Warren and Peter Szeredi, editors, *Logic Programming: Proceedings of the Seventh International Conference*, pages 579–597, 1990.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[IS98] Katsumi Inoue and Chiaki Sakama. Negation as Failure in the head. *Journal of Logic Programming*, 35(1):39–78, 1998.

[Lif94] Vladimir Lifschitz. Minimal Belief and Negation as Failure. *Artificial Intelligence*, 70:53–72, 1994.

[Lif96] Vladimir Lifschitz. Foundations of logic programming. In G. Brewka, editor, *Principles of Knowledge Representation*, pages 1–57. CSLI Publications, 1996.

[Llo87] John W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, New York, second edition, 1987.

[LW92] V. Lifschitz and T.Y.C. Woo. Answer Sets in General Nonmonotonic Reasoning (Preliminary Report). In C. Rich B. Nebel and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 603–614. Morgan Kaufmann, 1992.

[Min82] Jack Minker. On Indefinite Data Base and the Closed World Assumption. In D. Loveland, editor, *Proc. 6th Conf. on Automated Deduction (CADE'82)*, pages 292–308, LNCS 138, New York, 1982. Springer.

[Sak89] Chiaki Sakama. Possible Model Semantics for Disjunctive Databases. In *First International Conference on Deductive and Object-Oriented Databases (DOOD'89)*, pages 369–383, North-Holland, 1989.

[Wag94] Gerd Wagner. *Vivid Logic. Knowledge-based reasoning with two kinds of negation (Lecture Notes in Artificial Intelligence 764)*. Springer-Verlag, Berlin, 1994.