What's in a model?
Epistemological analysis of Logic Programming

Marc Denecker

Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, 3001 Heverlee, Belgium
`marcd@cs.kuleuven.ac.be`
http://www.cs.kuleuven.ac.be/ marcd

Abstract. The paper is an epistemological analysis of logic programming and shows an epistemological ambiguity. Many different logic programming formalisms and semantics have been proposed. Hence, logic programming can be seen as a family of formal logics, each induced by a pair of a syntax and a semantics, and each having a different declarative reading. However, we may expect that (a) if a program belongs to different logics of this family and has the same formal semantics in these logics, then the declarative meaning attributed to this program in the different logics is equivalent, and (b) that one and the same logic in this family has not been associated with distinct declarative readings.
In the current state of the art, neither (a) nor (b) holds. The paper investigates the causes and the consequences of this phenomenon and points out some directions to overcome the ambiguity.

1 Introduction

In the early 70-ties, the area of Knowledge Representation in AI was the scene of a lively debate between logical approaches versus non-logical approaches such as semantic networks. In 1975, a paper "*What's in a link?*" by Woods [22] had a significant impact on this debate. In this paper, he pointed out that semantic nets were *epistemologically ambiguous* in the sense that "*the same semantic network notations could be used by different people (or even by the same person at different times for different examples) to mean different things*". If there is anything that we should request of a logic or a knowledge representation language, then it is that its expressions have a clear and non-ambiguous meaning. Given an expression or theory in such a language, it should be clear what information it expresses about the real world. Woods study was received as a strong argument in favor of the logical approach to knowledge representation and as a decisive argument in favor of using formal semantics for disambiguating the meaning of knowledge representation languages [12].

This study is an epistemological investigation of Logic Programming (LP) and investigates its *declarative reading(s)*. The notion of declarative reading of a theory is used here to refer to the intuitive meaning of the theory, i.e. the information on the external problem world that a human expert extracts from the theory. Although this declarative reading is not a formal object, it is (or ought to be) determined by the formal semantics. I will argue that at the epistemological level, LP and its extensions show the same type of epistemological confusion and ambiguity as occurred in the area of semantic nets in the early 70ties.

The area of logic programming offers a complex landscape and many different extensions and semantics have been presented. Each pair of formal syntax and semantics in the LP family can be expected to have its own declarative reading, thus giving rise to a complex landscape of different declarative interpretations and views. However, this is not the ambiguity problem that I am referring at. There is a deeper problem.

There is a substantial overlap between the different LP logics. For example, for acyclic normal programs, it is well known that completion semantics, the stable semantics and the well-founded semantics coincide [1]. For stratified programs, stable, well-founded and perfect semantics coincide. In general, we should expect that:

(a) the declarative reading underlying the different semantics are *equivalent* for programs for which different semantics coincide;
(b) each logic consisting of a pair of syntax and semantics in this family has a unique declarative reading. If in one way or the other, different readings have been associated to the same pair of syntax and semantics, then these readings must be equivalent in some deep sense.

The epistemological ambiguity of logic programming is that neither (a) nor (b) holds. It is easy to show that virtually all programs, including those for which all the above mentioned semantics coincide, have been assigned several different declarative readings. This is an ambiguity of an analogous kind as in the context of semantic nets in the early seventies.

As shown in section 2, this ambiguity can be pinpointed in a formal way, by comparing different formal embeddings of logic programming in other logics. Section 3 investigates how this ambiguity can arise despite the existence of formal semantics.

The observation that logic programming has multiple interpretations has been made before. In [11],Gelfond and Lifschitz observe that extended logic programs without classical negation (under answer set semantics) and general logic programs (under stable semantics) are formally indistinguishable (same syntax and formal semantics) but that "*in spite of this, there is a semantic difference between a set of rules viewed as a general program and the same set of rules viewed as an extended program*". What they say here is that although general logic programming and extended logic programming without classical negation are formally identical, they have different declarative readings and should be considered as different logics. Section 4 discusses how the ambiguity phenomenon appears in the LP literature. Section 5 investigate some of the consequences. Finally, section 6 discusses some approaches for a more solid epistemological foundation of LP, and points out some directions to resolve the epistemological ambiguity.

2 The ambiguity formally demonstrated

A number of different logic programming formalisms and different semantics have been proposed. In this section I will focus on the standard formalisms of propositional definite and normal logic programming [14]. A normal or general logic program is a set of rules `p :- q`$_1$`,..,q`$_m$`, not q`$_{m+1}$`,.., not q`$_n$, where `p` and `q`$_i$ are (propositional) atoms. A definite program is a normal logic program without negative literals (m=n). An extended logic program consists of rules where `p` and `q`$_i$ are atoms or classically negated atoms $\neg$`q`. For these formalisms, a number of different model semantics have been proposed. The most important ones are the completion semantics [3], the stable semantics [10] and the well-founded semantics [21] for normal programs; I will also consider the least Herbrand model semantics for definite logic programs [20]. Some familiarity of the reader with these semantics and their interrelations and with classical logic, default logic and autoepistemic logic, will prove useful but most arguments presented below can be understood without deep acquaintance of the mathematical details of these systems.

The meaning of Logic Programming has often been defined through embeddings in other logics, in particular in (propositional) classical logic (CL), default logic [19] and autoepistemic logic [17]. To understand (some of) the multiple views on logic programming, we can compare the embeddings of LP in these different logics.

There is good formal ground for such comparisons, both in terms of possible state semantics of these logics and in terms of belief sets, i.e. first order or modal theories closed under entailment. The deductive closure of a first order theory, an *extension* of a default theory and an *expansion* of an autoepistemic theory[1] are belief sets that can be seen as the representation of the belief of the knowledge representing expert. The models of these belief sets represent the possible states according to the expert's belief. Thus, these objects can be compared in a meaningful way.[2]

We consider the three oldest and most influential embeddings of LP. The first embedding is the Clark completion [3], which defines the declarative reading of a normal program through a set of first order equivalences, called completed definitions. The *completed definition* of an atom `p` occurring as the head of rules `p :- L`$^i_1$`,..,L`$^i_{n_i}$ ($1 \leq i \leq m$), is the following first order equivalence:

$$p \leftrightarrow (L^1_1 \wedge .. \wedge L^1_{n_1}) \vee ... \vee (L^m_1 \wedge .. \wedge L^m_{n_m})$$

The completion of a propositional program P is the first order theory consisting of the set of completed definitions of all symbols and will be denoted $comp(P)$.

In [8], Gelfond proposed to interpret negation as failure literals `not p`, which are interpreted by the Prolog system as "p cannot be proven", by epistemic literals "I do not know p". To formalise this declarative reading, he proposed an embedding of logic programs into autoepistemic theories in which negation as failure literals `not p` are mapped to epistemic literals $\neg Kp$. In this embedding, a logic programming rule:

```
   p :-  q, not r
```

[1] An expansion contains not only first order formulas but also modal formulas. However, these can be computed from the first order formulas in it. For this comparison, I will ignore them.

[2] A first order theory T always defines a unique belief set $Cn(T)$. Default and autoepistemic logic are *indexical* in the sense that they have expressions referring to the agents own beliefs. As a consequence, they sometimes cannot be assigned a belief set or they may have multiple belief sets, each being a candidate for representing the experts belief. This distinction is not important for the sake of the argument in this section, since the examples that we will consider have a unique belief set.

is interpreted as the following AEL formula:

$$p \leftarrow q \wedge \neg Kr$$

In the sequel, I will denote the mapping of a logic program P as $ael(P)$. In the original paper on the stable models [10], this embedding is used to explain the stable semantics. In particular, it was shown that the stable models of a program P correspond exactly to the sets of atoms in the *autoepistemic expansions* of $ael(P)$.

In [15], Marek and Truszczynski proposed a third embedding, of logic programming into default logic. This embedding was later generalised by Gelfond and Lifschitz in [11] to the case of answer set programming. The embedding maps a rule

```
p :-  q, not r
```

to the default:

$$\frac{q : \neg r}{p}$$

Let us denote the embedding of a program P by $dl(P)$. Marek and Truszczyński showed that there is a one to one correspondence between stable models of P and extentions of $dl(P)$: each extention of $dl(P)$ is of the form $Cn(M)$ with M a stable model and vice versa, for each stable model M, $Cn(M)$ defines an extension of $dl(P)$.

The three different embeddings are non-equivalent and give different meaning to the rule operator (equivalence, material implication or inferent rule) and the negation as failure (classical negation, or modal epistemic negation). The difference appears in almost every logic program, even the most simple ones without negation and where all main LP model semantics coincide. It is easy to demonstrate this formally. Consider the case of definite programs and as an illustration, take the following program:

$$P_1 = \{\texttt{p :- q}\}$$

The first point is that P_1 is a non-recursive definite program. For such programs, all semantics coincide. The empty set $\{\}$ is the unique least model, the unique model of the completion, the unique stable model and the unique well-founded model of P_1.

Now, let us compare the meaning of P_1 as expressed by the different embeddings on which these model semantics are based. The following table presents the different embeddings, the belief sets and the possible states:

$comp(P)$	$ael(P_1)$	$dl(P_1)$
$\{q \leftrightarrow false,$ $p \leftrightarrow q\}$	$\{p \leftarrow q\}$	$\left\{\frac{q:}{p}\right\}$
$Cn(\{\neg p, \neg q\}$	$Cn(\{p \leftarrow q\})$	$Cn(\{\})$
$\{\}$	$\{\}, \{p\}, \{p, q\}$	$\{\}, \{p\}, \{p, q\}, \{q\}$

On the one hand, we observe that the least model, the model of the completion, the stable and the well-founded model of P_1 coincide. On the other hand, we observe that the three embeddings assign a different belief set and a different set of possible states to P_1. Consequently, P_1 is an example of a program for which all formal model semantics coincide, but for which the declarative readings induced by the different embeddings differ. So it illustrates that point (a) of the introduction does not hold.

P_1 also illustrates that ael and dl assign a different meaning to logic programs, despite the fact that they both induce the stable semantics. So, this example illustrates that point (b) of the introduction does not hold.

The above example is just one illustration of a large class of programs showing the ambiguity problem. Note that P_1 is a definite logic program, and hence, these different views on the meaning of a logic program occur even in the absence of negation as failure.

We can conclude that the different embeddings give different meaning to the same logic programs. Whether this is an ambiguity of the same order as that of semantic nets in the early seventies, depends on whether these different meanings have been mixed up in the LP community, which is discussed in section 4. But first we look at how the ambiguity could arise.

3 Ambiguity: how can it arise?

Unlike semantic nets in the middle seventies, the ambiguity of LP arises in the context of logics with formal model semantics. As illustrated above, *formal semantics do not seem to guarantee a non-ambiguous declarative reading?* How is it possible that the embeddings differ while the induced models coincide?

In a nutshell, a formal semantics is not more than a precise mathematical theory which associates formal semantical objects (i.e. models, usually) to a logic expression. But formal semantics cannot prescribe how human experts should interpret what these semantical objects say about the real problem world; what a model means is informal and may be subjective. Two human experts who give different interpretations to what the models of a logic expression mean, will assign different meaning to the expression.

This phenomonon occurs in logic programming. In each of the embeddings considered in the previous section, a different epistemological role is assigned to a "model". Under the completion, in the classical logic convention, a model represents a *possible state of the world*, describing the collection of objects that exists in this state, their relationships and the functions. Under the default embedding *dl*, a stable model (or more general, an answer set) M is a first order logic representation of a possible belief state of the expert and $Cn(M)$ is the corresponding belief set. In the autoepistemic embedding ael, a stable model is simply the set of atoms that are believed in one of the possible belief states of the expert.

The same mathematical structures are used to describe very different characteristics of the expert knowledge. For example, the fact that the unique model of $comp(P_1)$ is $\{\}$ means that the world is known to be in the state in which both p and q are false. The fact that the stable model is $\{\}$ means in the autoepistemic view that nor p nor q are known to be true by $ael(P_1)$. And in the default view $dl(P_1)$, it means that the empty theory is believed, i.e. the expert knows nothing and all he believes are the tautologies.

The bottom line is that the informal notions of declarative reading of a formal logic and the epistemological role of its semantic primitives are tightly connected. Only if we fix the meaning of the semantic primitives, the formal semantics specifies a declarative reading of the logic.

4 Ambiguity in the literature

In the context of classical logic, there is a *convention* that is quasi universally followed, namely that a model represents a possible state of the world. In logic programming, this convention is not followed. This is a potential source of confusion. Below is a brief overview of whether and how epistemological foundations of logic programming are discussed in the literature.

In the papers introducing the main semantics of logic programming [20, 3, 2, 21], the epistemological role of the models is not discussed. It can be that the authors had only the intention to define a formal semantics and not a declarative reading, or it may be that they had in mind the standard convention of interpreting a model as a possible state of the world, but we cannot be sure. One exception is Gelfond and Lifschitz's original paper on stable semantics [10], which explicitly state that stable models represent *possible states of beliefs that a rational agent might hold"*. As appears from [11], they later left this view in the context of general logic programs but kept it in the context of extended logic programs and answer set programming.

In [11], the difference between *general logic programs* and *extended logic programs (without* $\neg$*)* is explained in terms of different interpretations of a stable model. Gelfond and Lifschitz state that *the absence of an atom* A *in the stable model of a general program represents the fact that* A *is false; the absence of* A *and* $\neg A$ *in the answer set of an extended program is taken to mean that nothing is known about* A. What this means is that in general logic programming, a stable model represents a possible state of the world, whereas in answer set programming, it represents a first order theory of believed literals (exactly as under *dl*).

More recently, the view of answer set programming seems to evolve towards a computational paradigm, in which epistemological issues are ignored. In [16], Marek and Truszczyński investigate the stable logic progamming paradigm from a computational and complexity perspective and ignore epistemological aspects of the semantics; they do not discuss the epistemological role of stable models nor the declarative reading induced by the stable semantics. In [18], Niemelä mentions that *logic programs can be seen as a special case of autoepistemic logic* and as default theories and refers to stable models as *solution sets*. In these papers, the distinction between general logic programming and answer set programming is fading and the view on what a model means is not longer considered to be an issue; instead a stable model or an answer set represents a "solution" to a "problem". These terms have no epistemological connotation and it is impossible to know what a stable logic program or answer set program means. Some counterweight is found in [9] which distinguishes general logic programs from answer set programs and discusses how to approximate the first by the latter.

Abductive logic programming is another extension of logic programming. In [13], Kakas, Toni and Kowalski define it from an inferential point of view, as an extension of of logic programming to perform abductive reasoning. Formal semantics are specified implicitly through a framework explaining how in any logic programming semantics can be extended to the case of ALP. Until recently, no efforts had been made to explain the epistemological foundations of this logic, and this domain inherits the ambiguity problems of LP.

In conclusion, there exist multiple declarative readings of logic programming. Gelfond and Lifschitz have explained the distinction in some depth, but epistemological issues have mostly been ignored or confused. The picture of logic programming's epistemology is a blurred one.

5 Consequences

The declarative reading and, connected, the view on the semantical primitives, are informal notions but they play a fundamental role in the goals and applications of declarative logic in AI. Logic is often praised for its precision as a specification and a knowledge representation language. It is seen as a precise language for communication of information between human experts. Evidently, this precision depends on the extent to which different human experts interpret a logic theory in the same way, that is, assign the same declarative reading to it.

Also the KR methodology of a declarative logic is based on its declarative reading. The basic methodological principle of knowledge representation in declarative logic is that the human expert describes the world by writing formal axioms or theories that are *true* in the external problem world. Different declarative readings require different methodologies. The methodologies of answer set programming and general logic programming are indeed very different.

Below, I discuss two consequences of the multiple interpretation of logic programming.

An example of the kind of confusion that may arise due to the epistemological ambiguity is the following. In LP, there exist many mathematical results relating different model semantics. Because of the different views on what a model is, extreme caution is needed when interpreting these results at the epistemological level. Comparing first order theories of literals, sets of believed atoms and possible worlds is comparing apples and oranges. These mathematical results are potentially very misleading and may be simply meaningless. For example, in [10] it is shown that the set of stable models is a subset of the set of models of the completion. It is tempting to conclude from this that the default reading or the autoepistemic reading of a logic program is stronger, i.e. has larger belief sets, than the completion semantics. This is a wrong conclusion. Actually the program P_1 in the previous section is an example showing that the contrary often happens. The belief represented by $dl(P_1)$ is a strict subset of the belief represented by $ael(P_1)$ which in turn is a strict subset of the belief set represented by $comp(P_1)$. In this example and in many others the completion semantics is stronger than the default reading. On the other hand, the meaning of a set of normal rules interpreted under completion semantics is indead weaker than the same set of rules seen as a general logic program (under stable semantics).

In LP and the wider area of nonmonotonic reasoning, there are plenty of mathematical results relating different systems and semantics, but very little attention is spent to explain what these mathematical relationships mean at the epistemological level.

Another issue is the nature of the *negation as failure*. It is generally assumed that negation as failure is not classical negation. This view is supported by the embeddings of LP in default and autoepistemic logic. But these embeddings only apply for answer set programming, and not for, say general logic programming.

A closer look at Clark's completion transformation shows that negation as failure is translated to classical negation. In the completion semantics, negation as failure is definitely classical negation; it is the LP rule operator that is assigned a non-classical meaning. In general logic programming (under stable semantics) and completion semantics, models are viewed in the same way, as possible states. Stable models of a general logic program are models of the completion. For an important class of programs, the inverse holds. Such programs, whether interpreted as general logic programs semantics or under the completion semantics, are mathematically and epistemologically equivalent. In the light of this, one wonders if negation as failure in general logic programming is not classical negation after all.

In [11], the meaning of a general logic program is explained in terms of the corresponding extended logic program augmented with the closed world assumption for all predicates. However, in the current practice of general logic programming, this explanation is not satisfactory. General logic programs are frequently used to model domains where the human expert has incomplete knowledge. For example, in the context of a planning problem, there is knowledge about initial state, effects of actions and goal state, but not about what actions actually occur. In such domains the closed world assumptions are invalid. The issue of the nature of negation as failure remains partly unresolved.

6 In search of a solid epistemology for LP

Originally, logic programming was viewed as the Horn clause sub-formalism of classical logic. This view broke down almost immediately when the *negation as failure* inference rule was introduced. On the one hand, this inference rule is *unsound* with respect to the classical logic view of a logic program. On the other hand, negation as failure derived conclusions with a strong *common sense* appeal and turned out to be very useful and natural in many practical situations. This was the motivation for the search for alternative views and semantics of logic programming.

One intuition is to interpret a negation as failure literal `not p` as an epistemic statement of the kind "*I do not know* `p`", or similarly "*I cannot infer* `p`", or "*it is consistent to assume that* `p` *is false*". This approach has been followed in answer set programming.

A second view is to interpret a logic program as a *definition*. This intuition was first used by Clark [3] in the completion semantics. As mentioned in the previous section, negation as failure in this view is really standard classical negation but LP's rule operator is not material implication. The completion semantics has a big disadvantage that it does not correctly formalise the intended meaning of recursive logic programs representing inductive definitions (e.g. the transitive closure program). But other semantics, in particular the perfect model semantics and well-founded semantics solve this problem. In [4–6], we argued that logic programming under well-founded semantics can be seen as a logic of a very general kind of definitions, including many types of mathematical inductive definitions. In [5], a logic, called ID-logic, is defined as an integration of classical logic assertions and inductive definitions. ID-logic formally extends Abductive Logic Programming and induces a declarative reading for ALP as a logic of definitions and classical logic assertions.

The source of LP’s ambiguity problem is the fact that these fundamentally different intuitions lead to the same or very similar formal semantics but in which the epistemological status of a model is different.

There is no “right” view on logic programming. Both intuitions can be the basis of a different epistemology for LP (which to some extend have been elaborated already). The two views are both consistent, natural and have their own class of applications. Prototypical examples of inductive definitions are programs such as `member, append, delete,..` and the transitive closure program. Prototypical examples of the other view are defaults and rules referring to knowledge or provability as in:

```
presumed_innocent :- not guilty
```

Both views lead (and have led already) to different logics, in which the epistemological role of the semantic primitives has been made explicit. It is easy to avoid the ambiguity in the context of knowledge representation and declarative semantics; it suffices to state explicitly in which of these logic one works. Then it is clear what is the role of the semantic primitives.

There remain important philosphical questions about the meaning of the logical connectives (negation as failure, the rule operator and disjunction in the head) in the different logics of logic programming. For example, one open question, stated in the previous section, is whether general logic programming imposes closed world assumption and whether its negation as failure is really modal negation as suggested by Gelfond and Lifschitz.

7 Conclusion

One aim, maybe the main aim, of formal semantics of a logic is to clear out the meaning of its expressions. I showed that in the logic programming community, there remains considerable vagueness and ambiguity about the meaning of programs and the logical connectives, despite many efforts on the formal semantics.

Epistemological clarity is a *sine qua non* for declarative logic. If we want to advocate the use of logic programming as a knowledge representation language to the broader knowledge representation and AI community, we will have to explain in clear and precise terms what knowledge is represented in our logic. If we want to understand the position of logic proramming in the spectre of logics, we will have to explain what the logical connectives mean and how they relate to connectives in other logics.

The ambiguity of logic programming is a concern for anyone who wishes to view logic programming and its extensions as declarative logics. Logic programming must be to split out in a number of logics with a clear declarative reading and a well-motivated precise formal semantics. This will greatly improve our understanding of the position and contributions of logic programming in logic and knowledge representation and I believe that in the long run it will lead to a much desired simplification of the very complex landscape of logic programming.

Acknowledgements

This paper is an outcome of many discussions with many people over many years. In particular I wish to thank Michael Gelfond, Vladimir Lifschitz, Victor Marek, Eugenia Ternovska, Mirek Truszczynski and anonymous referees of previous versions of this paper.

References

1. K.R. Apt and M. Bezem. Acyclic programs. In *Proc. of the International Conference on Logic Programming*, pages 579–597. MIT Press, 1990.
2. K.R. Apt, H.A. Blair, and A. Walker. Towards a theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.
3. K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.
4. Marc Denecker. The well-founded semantics is the principle of inductive definition. In J. Dix, L. Fariñas del Cerro, and U. Furbach, editors, *Logics in Artificial Intelligence*, volume 1489 of *Lecture Notes in Artificial Intelligence*, pages 1–16, Schloss Daghstull, October 12-15 1998. Springer-Verlag.
5. Marc Denecker. Extending classical logic with inductive definitions. In J. Lloyd et al., editor, *First International Conference on Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 703–717, London, July 2000. Springer.
6. Marc Denecker, Maurice Bruynooghe, and Victor Marek. Logic programming revisited: logic programs as inductive definitions. *ACM Transactions on Computational Logic*, 2(4):623–654, October 2001.
7. S. Feferman. Formal theories for transfinite iterations of generalised inductive definitions and some subsystems of analysis. In A. Kino, J. Myhill, and R.E. Vesley, editors, *Intuitionism and Proof theory*, pages 303–326. North Holland, 1970.
8. M. Gelfond. On Stratified Autoepistemic Theories. In *Proc. of AAAI87*, pages 207–211. Morgan Kaufman, 1987.
9. M. Gelfond. Representing knowledge in a-prolog. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond; Essays in honour of Robert A. Kowalski, Part II*, number 2407 in Lecture Notes in Computer Science, pages 413–451. Springer Verlag, 2002.
10. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the International Joint Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.
11. M. Gelfond and V. Lifschitz. Logic Programs with Classical Negation. In D.H.D. Warren and P. Szeredi, editors, *Proc. of the 7th International Conference on Logic Programming 90*, page 579. MIT Press, 1990.

12. P. Hayes. In defense of logic. In *Proc. of the 5th IJCAI77*, 1977.
13. A. C. Kakas, R.A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.
14. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
15. V.W. Marek and M. Truszczyński. Stable semantics for logic programs and default reasoning. In E.Lust and R. Overbeek, editors, *Proc. of the North American Conference on Logic Programming and Non-monotonic Reasoning*, pages 243–257, 1989.
16. V.W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In K.R. Apt, V. Marek, M. Truszczynski, and D.S. Warren, editors, *The Logic Programming Paradigm: a 25 Years Perspective*, pages pp. 375–398. Springer-Verlag, 1999.
17. R.C. Moore. Semantical Considerations on non-monotonic logic. In *Proc. of IJCAI-83*, pages 272–279, 1983.
18. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.
19. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
20. M. van Emden and R.A Kowalski. The semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742, 1976.
21. A. Van Gelder, K. A. Ross, and J.S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.
22. W. Woods. What's in a link: Foundations of semantic networks. In D. Bobrow and A. Collins, editors, *Representation and understanding: Studies in cognitive science*. Academic Press, New York, 1975. Also in Brachman and Levesque, *Readings in Knowledge Representation*, Morgan Kaufman, 1985.