What's in a Model? Epistemological Analysis of Logic Programming

Marc Denecker
Departement Computerwetenschappen,
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee,
Belgium
email: marcd@cs.kuleuven.ac.be

Abstract

It is commonly believed that the meaning of a formal declarative knowledge representation language is determined by its formal semantics. This is not quite so. This paper shows an epistemological ambiguity that arises in the context of logic programming. Several different logic programming formalisms and semantics have been proposed. Hence, logic programming can be seen as an overlapping family of formal logics, each induced by a pair of a formal syntax and a formal semantics. We would expect that (a) each such pair has a unique declarative reading and (b) for a program in the intersection of several formal LP logics with the same formal semantics in each of them, its declarative reading is the same in each of them.

I show in this paper that 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.

Introduction

This paper is concerned with the use of logic programming (LP) languages for *declarative knowledge representation*, and with the *declarative reading* of LP languages in relation to the formal semantics. I will first try to clarify what I mean with these terms.

In the process of *declarative knowledge representation* (as viewed here in this paper) about some *domain of discourse*, a human expert, at some stage, will select a set of relevant basic properties that are known to hold in this domain and which he desires to represent in the formal language. To formalise these properties, he needs to design an appropriate vocabulary of so-called *non-logical* symbols to denote relevant objects and concepts in the domain. Once these symbols are chosen and their denotations clear and fixed, well-formed expressions of the language become meaningful statements about the domain in the mind of the human expert; a formal expression is then a statement of a certain *property*, which can be *true* or *false* in that domain. This property is what is called here the *declarative reading of the expression* (under the given denotations of the non-logical symbols). The experts goal is to construct a set of formal expressions whose declarative readings match the selected basic properties.

Not all formal languages aim to be used or can be used for declarative knowledge representation in the above sense. To be able to use a formal language as in the above scenario, the following question of epistemological nature must have an answer: *for any given domain of discourse and set of denotations of non-logical symbols in this domain, what is the property of the domain that is expressed by a formal expression of the language?* This epistemological question is an informal question and cannot be answered with absolute mathematical rigor. Yet, for a class of languages, the question can be answered with a very high degree of precision. The standard example of such a language is of course classical logic. For example, when using unary predicate symbols $human$, $male$ and $female$ to denote respectively humans, males and females, the declarative reading of the classical logic formula

$$\forall x(human(x) \rightarrow male(x) \vee female(x)) \qquad (1)$$

is the property that each human is male or female or both. This is a precise and objective property, which is *true* in the real world. A well-known historical example of an *epistemologically ambiguous* knowledge representation language, i.e. one whose expressions are interpreted differently by different human experts, was the formalism of semantic networks in the state of the art of the early seventies. (Woods 1975) pointed out 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*": for example, one network could be understood by some as the property that all telephones are black, by others that some telephone is black and by yet others as a definition of the concept of a black telephone.

Epistemological ambiguity interferes with the basic role of a declarative knowledge representation language: to be a precise and reliable tool to describe properties of a domain and communicate these amongst human (or non-human) experts. Indeed, a language can only play this role if the community that uses it is able to assign a common, unique, well-understood declarative reading to the expressions of the logic, in the context of any appropriate set of denotations of the non-logical symbols. A non-ambiguous declarative reading is also needed to develop a coherent declarative knowledge representation methodology. Consequently,

one of the main conclusions from semantic networks drawn by Woods and by the logic-based AI community (e.g. in (Hayes 1977)) was that the meaning of knowledge representation languages must be characterised in a formal mathematical way, by formal semantics. At present, many in the AI community seem to believe that a formal semantics disambiguates the meaning of a formal language: that it determines the declarative reading of the language, if not in an explicit then at least in an implicit way. A main point of this paper is that this is not so, and I will illustrate this point in the case of classical logic and logic programming.

This study is an epistemological investigation into logic programming. The area of logic programming is complex and several different semantics and extensions have been presented. The most common semantics are the completion semantics (Clark 1978), the well-founded semantics (Van Gelder, Ross, & Schlipf 1991), the stable model semantics (Gelfond & Lifschitz 1988) and its extension, answer set semantics (Gelfond & Lifschitz 1991). The extensions that concern us here are abductive logic programming (ALP) (Kakas, Kowalski, & Toni 1992) and answer set programming (ASP)(Gelfond & Lifschitz 1991). Logic programming is not one formal language but can be seen as a family of such languages, each induced by a pair of a syntax and a formal semantics.

These languages belong to the intersection of computational logic and knowledge representation logics. As computational logics, they serve to encode problems and to solve them by applying some standard inference mechanism. But they are also often viewed as *declarative* languages which can be used for knowledge representation and which are related to default and autoepistemic logic. Viewed from this perspective, the epistemological question arises: what is the declarative reading of logic programming? What is the meaning of the negation as failure and the rule operator? These questions have occurpied many researchers since the late seventies. In this paper, I do not aim to answer these questions. Rather, I want to pinpoint a problem in the logic programming community which causes an epistemological ambiguity of similar nature as occurred in semantic nets in the early seventies.

Evidently, we may not expect that the negation as failure symbol and the rule operator have the same meaning in each of the LP logics that are induced by a selection of syntax and formal semantics. On the contrary, each of these languages must be expected to have its own declarative reading, thus giving rise to a landscape of different interpretations and views. However, we might expect the following.

(a) We might expect that each LP logic in this family has a unique declarative reading. Even if we miss the words and concepts in natural language to give a very precise explanation of the intuitive meaning of the logical symbols in the logic (i.e. negation as failure, the rule operator, ...), we would expect that the formal semantics implicitly disambiguates the meaning of the logic expressions.

(b) There is a substantial overlap between the different LP logics. For example, for acyclic normal programs, completion semantics, the stable semantics and the well-founded semantics coincide (Apt & Bezem 1990). For stratified programs, stable and well-founded semantics coincide. If a program belongs to several LP logics and has the same formal semantics in them, then we might expect that its declarative readings in these different logics agree with each other.

The epistemological ambiguity of logic programming is that neither (a) nor (b) holds. There exist different views on the meaning of logic programs, including those for which all semantics coincide. Moreover, the same expressions under the same semantics have sometimes been interpreted in different ways. This is an epistemological ambiguity of an analogous kind as in the situation of semantic nets in the early seventies.

This phenomenon contradicts with the common belief that formal semantics disambiguates the meaning of a formal language. A formal semantics merely defines a formal relationship between expressions of a formal language and mathematical semantical concepts, e.g. structures and the $\models$ relation between structures and expressions. If the meaning of the semantical concepts is not clear and different human experts assign different epistemological roles to these semantical concepts, then they will disagree about the meaning of the expressions of the logic, even if they agree on the formal semantics. This point is illustrated in the next section where a non-standard declarative reading of classical logic is presented which is compatible with the standard semantics of classical logic.

The above observation shows the hart of LP's ambiguity problem. There is no clear, universally adopted convention of what a model means in the research community of LP, answer set programming (ASP) and abductive logic programming (ALP). This explains why different intuitions can and have been assigned to the same LP expressions, even to those for which all formal semantics coincide. The paper investigates the consequences and problems caused by the ambiguity and discusses general strategies to resolve the ambiguity.

An alternative interpretation of classical logic

The following table describes the standard meaning of the logical connectives and of the basic semantical concepts of model theory of classical logic:

formal concept	meaning
atom $p(x_1, \ldots, x_n)$	$(x_1, \ldots, x_n)$ belongs to 'p'
$\wedge$	conjunction
$\vee$	inclusive disjunction
$\neg$	negation
$\exists x(\psi[x])$	existential quantification
$\forall x(\psi[x])$	universal quantification
a structure M	M formally represents a state of the problem world
$M \models \psi$	ψ is true in M
a model M of ψ	M is a possible state according to ψ
$\psi \models \phi$	ψ entails ϕ (ϕ is true in all states where ψ is true)

Given a vocabulary of user defined symbols denoting objects and concepts in a domain of discourse, the above table tells a human expert how to interpret a formula in the domain of discourse. For example, under the obvious meaning of the symbols, the formula

$$\forall x(\neg person(x) \vee male(x) \vee female(x)) \qquad (2)$$

expresses that each person is male or female or both.

The question now is whether the formal semantics of classical logic really forces a rational person to interpret the logical symbols as specified by this table. Stated differently: could it be that an intelligent and rational person understands the formal semantics of first order logic, and yet, that this person interprets some or all logical connectives differently than in the above table? More precisely, is it possible that this person would understand for which pairs of structures M and formulas F the satisfaction relation $M \models F$ holds and still, would assign a different meaning to the logical connectives? The answer is "yes" if the person also assigns a different meaning to the semantical concepts. Below is an alternative table:

formal concept	meaning
atom $p(x_1, \ldots, x_n)$	$(x_1, \ldots, x_n)$ does not belong to 'p'
$\wedge$	inclusive disjunction
$\vee$	conjunction
$\neg$	negation
$\exists x(\psi[x])$	universal quantification
$\forall x(\psi[x])$	existential quantification
$M \models \psi$	ψ is false in M
a model M of ψ	M is an impossible state according to ψ
$\psi \models \phi$	ψ is entailed by ϕ

Using this table, any formula means exactly the opposite of its standard declarative reading. For example, the formula (2) means that *there exists a person who is neither male nor female*. Yet, this is consistent with the mathematical definition of the truth relation $\models$ because the latter relation now specifies falsity rather than truth.

This experiment shows that formal model semantics do not disambiguate the declarative reading of a formal language in an absolute manner. It shifts the question of the meaning of the logical connectives and formulas to the question of the meaning of the semantical concepts. Changing the meaning of the semantical concepts changes the meaning of the logical connectives, and as shown, the change can be drastic. Knowing which are the models of a formula does not tell us what this formula means. We also need to know what a model tells about the world, and this is not and cannot be formalised in the model theory.

Classical logic owes its reputation as a clear, precise, non-ambiguous knowledge representation language not only to its formal semantics but also to an implicitly but universally adopted *convention* about the meaning of two key semantical concepts: *a structure* is understood as a mathematical abstraction of a state of the problem domain and describes it as a domain of objects and a set of functions and relations in this domain; *the satisfaction relation* $\models$ relates structures with formulas that are *true* in it. As shown, one can think of other consistent interpretations of these concepts, but fortunately, nobody uses them.

The bottom line is that the informal notions of declarative reading of a formal logic and the epistemological role of its semantic concepts are tightly connected. Only if the research community uses a standard convention on the meaning of the semantic concepts, the formal semantics determines a declarative reading of the logic.

This observation has the following philosophical implication. It is well-known that there are many different views on what logic is. In AI, logic is often defined as a system consisting of a formal syntax and formal semantics. In the light of the observations in this section, we realise that such a system in principle has no fixed declarative reading. From a knowledge representation perspective, a more sensible view is that a logic is a precise formal language to represent knowledge. In this view, a logic cannot be a purely formal object in the sense that two logics may be formally indistinguishable or isomorphic and still have different informal declarative readings, each formalised by one and the same formal semantics but using different informal views of the semantical concepts. I believe this consequence must be accepted and that it makes more sense to view logic as a triple consisting of a formal syntax, an informal declarative reading and a formal semantics formalising this declarative reading under a specific interpretation of the formal semantical concepts.

Ambiguity of Logic Programming

There are several symptoms and observations that point to the ambiguity of logic programming. Below we discuss some of them. We use the following formal and informal concepts. A *possible (world) state* is a state of the world which is possible according to a given belief. In a context where the belief of an agent is not or only partially known, a *state of belief* of the agent is one of the possible beliefs this agent might have. A *belief set* T is a deductively closed set (T is its own deductive closure $Cn(T)$) which represents a state of belief of an ideally rational agent, that is an agent with unlimited deductive powers (if he believes ψ and ψ entails ϕ then he believes ϕ).

Gelfond and Lifschitz's original paper on stable semantics (Gelfond & Lifschitz 1988) takes a nonstandard view on a model and explicitly states that stable models represent *"possible states of beliefs that a rational agent might hold"*. Up to that moment, the epistemological role of a model in LP semantics (Least Herbrand (van Emden & Kowalski 1976), completion semantics (Clark 1978), perfect model semantics (Apt, Blair, & Walker 1988)) had not been explicitly mentioned. In (Gelfond & Lifschitz 1990), Gelfond and Lifschitz distinguished between *general logic programs*, in which a stable model represents a possible world state, which is the standard view on a model, and *extended logic programs* in which an *answer set* represents a first order theory consisting of literals and representing a possible state of belief of the human expert. But extended logic programs that do not use classical negation are formally identical to

general logic programs. Moreover, their answer sets are the stable models. This led to the remarkable situation that general logic programming and the subset of extended logic programming without classical negation are formally indistinguishable and yet, in the light of the previous section, these logics have different declarative readings. Gelfond and Lifschitz realised this and observed that "*there is a semantic difference between a set of rules viewed as a general program (under stable semantics) and the same set of rules viewed as an extended program (under answer set semantics)*" (Gelfond & Lifschitz 1990).

There are simple examples that illustrate the different declarative readings. For example, we might represent a definition of *dead* in terms of *alive* by the following logic program rule:

```
dead :- not alive                    (3)
```

If this is the only rule with `dead` in the head, it expresses that a person is dead if and only if he is not alive.

On the other hand, (Gelfond & Lifschitz 1990) uses the following singleton program to illustrate the meaning of the logical connectives `:-` and `not` in extended logic programming:

```
cross :- not train                   (4)
```

This program represents the knowledge of a self-reflective agent. The non-logical symbol $cross$ denotes that the agent crosses the railway, and $train$ means that a train is arriving. The operator `:-` is interpreted as material implication; the operator `not` corresponds to the modal operator "*I do not know that ..*" in autoepistemic logic. The intended meaning of the program is that *all the agent knows* is that "he crosses *if he does not know that* a train is arriving". Since this theory doesnot tell whether a train is arriving, the agent apparently does not know and hence, by application of the rule, he will cross.

These two rules have the same syntactic form but if we make abstraction of the different denotations of non-logical symbols, they represent different types of properties. For example, it is true that Bin Laden is dead if and only if he is not alive, but it is not true that he is dead if I do not know that he is alive. Vice versa, in a situation where an agent crosses the railway in firm but mistaken belief that no train is arriving, the classical logic equivalence $cross \leftrightarrow \neg train$ is violated but the autoepistemic sentence is true. In both statements, the basic connectives negation as failure `not` and rule operator `:-` have different meaning. In the definition, `not` has the standard modality of classical negation and `:-` is equivalence; in the railroad rule, `not` is a modal operator and the rule operator is material implication.

The unique model of the general logic program {`dead :- not alive`} is {`dead`}. This model is the unique possible state of the world, which means that the belief represented by the program is that `alive` is false and `dead` is true. The unique answer set of the extended logic program {`cross :- not train`} is {`cross`} which represents the belief that `cross` is true and nothing is believed about `train`. The difference in declarative readings of these two rules is not explained by different formal semantics but by the different epistemological roles assigned to the model.

Semantics of Logic Programming has often been analysed through embeddings in other logics, in particular in classical logic (CL), default logic (Reiter 1980) and autoepistemic logic (Moore 1983). Different embeddings give a different meaning to the rule operator, to negation as failure and hence, to logic programs. The difference appears in almost every logic program, even the most simple ones without negation on which 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:

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

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

Let us compare the meaning of P_1 as expressed by three different embeddings. The first one is the Clark completion (Clark 1978). In the case where P is a propositional logic program, the theory $comp(P)$ consists of the equivalences $p \leftrightarrow B_1 \vee \ldots \vee B_n$ where $\texttt{p :- B}_1, \ldots, \texttt{p :- B}_n$ is the set of rules with p in the head. The belief set of an agent whose knowledge is represented by P under this embedding is $Cn(comp(P))$. The second embedding $ael(P)$ maps a rule `p :- q, not r` to the autoepistemic formula $p \leftarrow q \wedge \neg Kr$ and was proposed first in (Gelfond 1987) and again in the original paper on the stable model semantics (Gelfond & Lifschitz 1988). The belief sets of P under this embedding are given by the *autoepistemic expansions* of $ael(P)$. In (Gelfond & Lifschitz 1988), it was shown that the stable models of a program P correspond exactly to the sets of atoms in the *autoepistemic expansions* of $ael(P)$. The third embedding $dl(P)$, proposed by (Marek & Truszczyński 1989), maps a rule `p :- q, not r` to the default $\frac{q : \neg r}{p}$. P's belief sets are given by the default extensions of $dl(P)$. There is a one to one correspondence between stable models of P and default extentions of $dl(P)$: each default extension of $dl(P)$ is of the form $Cn(M)$ with M a stable model of P and vice versa, for each stable model M, $Cn(M)$ defines an extension of $dl(P)$. In (Gelfond & Lifschitz 1991), this embedding was extended to extended logic programs.

Under each embedding, P_1 leads to a unique state of belief. The following table formally compares the belief sets of an agent believing $comp(P_1)$, $ael(P_1)$ and $dl(P_1)$, and the possible world states corresponding to these belief sets.

$P_1 = \{\texttt{p :- q}\}$	$comp(P_1)$
embedding	$\left\{ \begin{array}{l} q \leftrightarrow false, \\ p \leftrightarrow q \end{array} \right\}$
belief set	$Cn(\{\neg p, \neg q\})$
possible states	$\emptyset$

$P_1 = \{$`p :- q`$\}$	$ael(P_1)$
embedding	$\{p \leftarrow q\}$
belief set	$Cn(\{p \leftarrow q\})$
possible states	$\emptyset, \{p\}, \{p, q\}$

$P_1 = \{$`p :- q`$\}$	$dl(P_1)$
embedding	$\left\{ \frac{q:}{p} \right\}$
belief set	$Cn(\emptyset)$
possible states	$\emptyset, \{p\}, \{p, q\}, \{q\}$

The differences in meaning are explained by the fact that in each embedding, the model $\emptyset$ plays a different epistemological role. In $comp(P_1)$, the model $\emptyset$ of P represents the unique possible state of the world; in $ael(P_1)$ it represents the set of believed atoms, while in $dl(P_1)$, it is a propositional theory representing the belief of the agent.

The program 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 (b) 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 (a) of the introduction does not hold.

At this moment, the literature is vague and confusing about the epistemological role of models and answer sets of LP, ALP and ASP. As mentioned before, the early works on semantics of logic programming do not explicitly mention the epistemological role of a model. Therefore, in principle, we cannot know the declarative reading of logic programming under the least Herbrand, completion, perfect and well-founded semantics. Gelfond and Lifschitz (Gelfond & Lifschitz 1991) introduced different roles for a model but they were explicit about this and even renamed logics to distinguish formally indistinguishable systems by the epistemological role of models (confer *general* versus *extended logic programs*). Their example has not been followed in the LP community.

Abductive logic programming (Kakas, Kowalski, & Toni 1992) is defined as an extension of logic programming to perform abductive reasoning. Formal semantics are specified implicitly through a framework explaining how any logic programming semantics can be extended to the case of ALP. No commitments are made with respect to the epistemological role of models. Until recently, no efforts had been made to explain the epistemological foundations of this logic, and this domain inherits the ambiguity problems of LP.

Also in answer set programming, the role of answer sets has grown confused. Gelfond has continued to explain the view of answer sets as first order theories representing states of belief of a rational agent, and to make the distinction between answer set programs and general logic programs (see e.g. (Gelfond 2002) which studies how to approximate general logic programs by answer set programs). However, he is not followed by the ASP community. Marek and Truszczynski in (Marek & Truszczyński 1999) and Niemelä in (Niemelä 1999) explain that an answer set is *a solution* to a *problem*. This explanation does not tell us what is the epistemological role of an answer set, i.e. what an answer set tells about the real world or about the belief of the human expert or the agent whose beliefs are expressed in the program. If we analyse the applications presented in these papers and related ones (e.g. Hamiltonian path, colorability problems, n-queens, planning problems, computer configuration problems ...), then we see that the programs can be viewed naturally as incomplete descriptions of a problem world and that stable models represent possible states of the world satisfying the programs. In this respect, the approaches proposed in these papers are based on general logic programming rather than on answer set programming. For example, consider the Hamiltonian path program from (Marek & Truszczyński 1999):

$$\left\{ \begin{array}{l} \texttt{in(V1,V2) :- edge(V1,V2), not in}*\texttt{(V1,V2)} \\ \texttt{in}*\texttt{(V1,V2) :- edge(V1,V2), not in(V1,V2)} \\ \texttt{f :- in(V2,V1), in(V3,V1), not V2} = \texttt{V3, not f} \\ \texttt{f :- in(V1,V2), in(V1,V3), not V2} = \texttt{V3, not f} \\ \texttt{reached(a)} \\ \texttt{reached(V2) :- in(V1,V2), reached(V1)} \\ \texttt{f :- not reached(X), not f} \end{array} \right\}$$

The first two rules specify that in is a subset of the edges of a graph. The two next rules specify that in is a linear path through the graph (all stable models of a rule `f :- B, not f`, satisfy the body B). The next rules inductively define the reachability relation from an initial vertex a. The last rule specifies that each vertex should be reachable from a. This theory specifies that in is a Hamiltonian path in the graph $edge$. It's models represent possible states of the world. This is a general logic program[1]. In the recent book (Baral 2003), Baral identifies *general logic programming* (or *normal logic programming* under the stable semantics) with *AnsProg*, a sublogic of answer set programming without classical negation nor disjunction. The aforementioned semantical difference between stable models in both formalisms is not discussed. Even Lifschitz no longer explicitly mentions the epistemological role of answer sets or the distinction between general logic programs and answer set programs (e.g. in (Lifschitz 2002)). (**?**) investigates the representation of definitions in answer set programming. It is unclear now whether a rule `p :- not q` represents a rule of a definition or an autoepistemic statement.

Because the epistemological role of a model or an answer set is often not made explicit and because of the lack of a clear and widely accepted convention, in principle we can not know what a logic program, an abductive logic program or answer set program means.

Some Consequences and Problems

This section discusses some consequences and potential problems caused by the ambituity.

[1] It is worth mentioning that in (Marek & Truszczyński 1999), Marek and Truszczynski use the term *Stable logic programming*, not answer set programming. Only later, this approach was incorporated in answer set programming (see e.g. (Lifschitz 1999)).

What is negation as failure (NAF)? A widespread belief is that negation as failure is not objective negation as in classical logic but an epistemic operator "*it is consistent to assume* $\neg p$". The truth is that there are multiple consistent views on the meaning of logic programs and its basic connectives `not` and `:-`. In the context of answer set programming, negation as failure `not` can indeed be interpreted as a modal operator, as shown for example by the translation dl of extended logic programming into default logic. However, this interpretation should not be extrapolated to other views on LP in which a model is a possible state of the world, not even to general logic programming. This is most obvious in the completion semantics which maps negation as failure into classical negation. At present, we do not know what negation as failure means in general or stable logic programming.

Language extensions. Languages are extended on the basis of their declarative reading. Language extensions that make sense in one declarative reading do not make sense in others. For example, if negation as failure in general logic programming would turn out to be classical negation, then it would make no sense to extend this formalism with classical negation.

Methodology and teaching. The multiple declarative readings complicate the development of a methodology. The combination of vagueness about the epistemological role of semantic concepts, the presence of multiple intuitions and declarative readings and the complexities of the programming methodology form a perfect blend to confuse our students. For example, the definition of dead in terms of alive can be represented by the rule (3)

```
dead :- not alive
```

or alternatively, if classical negation can be used:

$$\left\{ \begin{array}{l} \texttt{dead :- } \neg\texttt{alive} \\ \neg\texttt{dead :- alive} \end{array} \right\}$$

In the area of answer set programming, both styles have been used. Each style has its own applications and limitations. Moreover, these styles should not be mixed. As an illustration, assume a student has an assignment to represent that *all an agent knows is that he crosses the railroad if he does not know there is a train and that it is safe to cross iff there is no train*. The second part of this assignment is a definition. In the spirit of (**?**) (and similar as in the $dead$ and $alive$ example), he might be tempted to solve this assignment by the program P_{safe} consisting of the rule (4) and one additional rule:

```
safe :- not train                    (5)
```

After all, each of the rules correctly represents half of the assignment. Unfortunately for the student, this is a wrong answer for the simple reason that a model cannot play the role of a possible world state and of a set of believed formulas at the same time. Since the agent has no knowledge about the train, he crosses. So the belief set of the agent is $Cn(\{cross, safe \leftrightarrow \neg train\})$. The unique stable model of P_{safe} is $\{safe, cross\}$. This is not the unique possible state of the world ($\{cross, train\}$ is another possible state) and it is not the set of believed formulas of the agent.

I do not claim that programs mixing up different declarative readings actually occur in the LP literature (probably not, since they yield wrong answers). But I believe that the existence of the different declarative readings and the vagueness that surrounds them, complicates the development of a consistent, comprehensible knowledge representation methodology and creates confusion amongst students, users and researchers.

Comparing formal semantics. In LP, there exist many mathematical results relating different model semantics. What is lacking almost completely in the LP literature is what these mathematical results mean at the epistemological level. Extreme caution is needed here. Comparing sets of atoms representing possible states, sets of believed atoms or first order theories, is comparing *apples and oranges*. For example, it is well-known 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 than the completion semantics (in the sense that more formulas are believed and less world states are possible), but this is a wrong conclusion, in a similar way as it is wrong to conclude that 2kg is less than 3 pounds. Actually the program P_1 in the previous section is an example of the contrary: $dl(P_1)$ is weaker than $ael(P_1)$ which is weaker than $comp(P_1)$.

Explaining LP to the outside AI-world. Logic programming and its extensions are quite widely used as tools for *implementing* AI-applications or building rapid prototypes. However, as long as we cannot resolve the vagueness about and confusion of its meaning, I believe that LP will not and should not be accepted as a declarative knowledge representation language in the larger AI and KR community[2].

A strategy for resolving the ambiguity

In computational logic, logic is often viewed as a formal language to encode problems and to solve them using inference engines. Extracting the answer to the problem from the computed semantical concepts of a theory is considered to be the responsibility of the human expert. This means that the epistemological role of semantical concepts, that is, the way they are related to the problem domain and to the knowledge of the human expert, is left as a design choice to the human expert. But as we saw in the case of classical logic, it is impossible then to associate a declarative reading to connectives and expressions of the logic and to develop a methodology on the basis of the declarative reading.

Declarative knowledge representation, as meant in this paper, is not the same as *encoding problems in a formal language*. One represents knowledge by writing formal expressions with declarative readings that are *true* in the domain

[2] In the past, it has been one of my own research goals to demonstrate the suitability of *abductive logic programming* for knowledge representation. At some point I realised the futility of such efforts as long as we are unable to explain what kind of knowledge can be represented by *abductive logic programs*.

of discourse. To be able to do this, a knowledge representation language must have a unique, precise, well-understood declarative reading.

In many theoretical studies of a formal system, e.g. development of inference methods, proof of their correctness, complexity of such algorithms and of formally defined computational problems within the logic, formal analysis of relationships with other logics, etc, informal aspects such as declarative reading and epistemological role of semantic concepts are immaterial. However, in those situations where the meaning of the logical connectives and expressions matter, the epistemological role of semantical concepts should be clear:

- when explaining the intuitions underlying formal semantics
- when discussing the meaning of the formal connectives
- when explaining methodology
- when presenting examples and applications.

In the present state of affairs, logic programming and its extensions, viewed as knowledge representation languages, are ambiguous. The following strategy can resolve the ambiguity problem.

As a first step, the community should develop a rigourous and systematic discipline of being precise about the epistemological role of semantical concepts in each application and experiment and in each discussion of the meaning of the logical symbols.

In a second step, the area should "install" a fixed and clear convention of how to interpret the semantical concepts. If multiple declarative readings exist, the formal language should be split up and renamed so that the ambiguity disappears (as was done by Gelfond and Lifschitz in the case of *general logic programs* and *extended logic programs*). The area should monitor that this convention is obeyed in examples and applications.

But to understand the use of a formal language as a knowledge representation formalism, to develop a methodology and to be able to teach it to students, more is needed than just a formal semantics and a clear convention about the meaning of the semantical concepts. We need to study the meaning of its connectives and the declarative reading in a sufficiently large class of expressions. Such epistemological studies could consist of the following elements:

- "Meaning preserving" transformations to or from other logics with a well- or better understood declarative reading. Examples in LP are the Clark completion embedding and the embedding dl of extended logic programming in default logic.
- Arguments from first principles. Frequently, this is done by identifying some informal linguistic construct and arguing that the formal logic models this construct correctly. An example in the context of LP is found in (Denecker 1998; Denecker, Bruynooghe, & Marek 2001), which points to nonmonotone forms of inductive definitions in mathematics and argues that logic programming under the well-founded semantics can be seen as a logic of such inductive definitions. The epistemological role of a well-founded model is a possible state of the world.
- Illustration of different aspects of knowledge representation in the logic in many different examples and applications.

Presently, the three best understood declarative views on logic programming are:

- logic programming and abductive logic programming under well-founded semantics as a logic of nonmonotone inductive definitions (Denecker, Bruynooghe, & Marek 2001; Denecker & Ternovska 2004b; 2004a);
- extended logic programming as a sublogic of default logic based on dl (Marek & Truszczyński 1989; Gelfond & Lifschitz 1991) or, equivalently, as a sublogic of autoepistemic logic under the stable semantics (Denecker, Marek, & aw Truszczyński 2003);
- answer set programming as an epistemic logic to encode possible belief states of an artificial rational reasoner (Gelfond & Leone 2002).

The declarative reading of other extensions such as general or stable logic programming (Marek & Truszczyński 1999) are currently not well-understood. At this moment, we are still lacking a good understanding of the differences and relationships between all these logics and of their domains of application.

Conclusion

The ambiguity of LP boils down to the fact that the same rules and the same programs can be and have been assigned different meanings. Despite the fact that LP logics have formal semantics, this is an epistemological ambiguity of the same kind as that of semantic nets in the early seventies. This paper is an analysis of the causes of this ambiguity and presents a strategy to remediate the problem. In particular I have argued that formal semantics, also in the context of classical logic, do not disambiguate the declarative reading of a formal language in an absolute way. What is needed more is a clear convention on the epistemological role of the semantical primitives. It is the lack of such a convention that causes LP's epistemological ambiguity.

This paper has articulated a particular view on *declarative knowledge representation*, which I believe is quite common in the logic-based AI community. To be able to use a formal language for this sort of knowledge representation, a unique, precise, well-understood declarative reading of the language is a *sine qua non.* Although I argued that logic programming in the current state of the art, does not satisfy this criterion, I believe that LP extensions have a great potential use for knowledge representation and declarative problem solving, if we are able to explain its declarative reading in clear and precise terms. If we succeed in this, it will greatly improve our understanding of the position and contributions of logic programming in logic and knowledge representation and it may lead to a much desired simplification of the 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, Jose Alferes, Maurice Bruynooghe, Danny De Schreye, Vladimir Lifschitz, Victor Marek, Luis Pereira, Eugenia Ternovska, Mirek Truszczynski and many others, including the anonymous referees of this paper and of several older versions of it.

References

Apt, K., and Bezem, M. 1990. Acyclic programs. In *Proc. of the International Conference on Logic Programming*, 579–597. MIT Press.

Apt, K.; Blair, H.; and Walker, A. 1988. Towards a theory of Declarative Knowledge. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann. 89–148.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Clark, K. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Databases*. Plenum Press. 293–322.

Denecker, M., and Ternovska, E. 2004a. Inductive Situation Calculus. In *Proceedings of Ninth International Conference on Principles of Knowledge Representation and Reasoning, Delta Whistler Resort, Canada*. URL = http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=41085.

Denecker, M., and Ternovska, E. 2004b. A logic of nonmonotone inductive definitions and its modularity properties. In Lifschitz, V., and Niemelä, I., eds., *7th International Conference on Logic Programming and Nonmonotonic Reasoning*.

Denecker, M.; Bruynooghe, M.; and Marek, V. 2001. Logic programming revisited: logic programs as inductive definitions. *ACM Transactions on Computational Logic* 2(4):623–654.

Denecker, M.; Marek, V.; and aw Truszczyński, M. 2003. Uniform semantic treatment of default and autoepistemic logics. *Artificial Intelligence* 143(1):79–122.

Denecker, M. 1998. The well-founded semantics is the principle of inductive definition. In Dix, J.; Fariñas del Cerro, L.; and Furbach, U., eds., *Logics in Artificial Intelligence*, volume 1489 of *Lecture Notes in Artificial Intelligence*, 1–16. Schloss Daghstull: Springer-Verlag.

Gelfond, M., and Leone, N. 2002. Logic programming and knowledge representation – a-prolog perspective. *Artificial Intelligence* 138(1-2):3–38.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of the International Joint Conference and Symposium on Logic Programming*, 1070–1080. MIT Press.

Gelfond, M., and Lifschitz, V. 1990. Logic Programs with Classical Negation. In Warren, D., and Szeredi, P., eds., *Proc. of the 7th International Conference on Logic Programming 90*, 579. MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 365–387.

Gelfond, M. 1987. On Stratified Autoepistemic Theories. In *Proc. of AAAI87*, 207–211. Morgan Kaufman.

Gelfond, M. 2002. Representing knowledge in a-prolog. In Kakas, A., and Sadri, F., eds., *Computational Logic: Logic Programming and Beyond; Essays in honour of Robert A. Kowalski, Part II*, number 2407 in Lecture Notes in Computer Science. Springer Verlag. 413–451.

Hayes, P. 1977. In defense of logic. In *Proc. of the 5th IJCAI77*.

Kakas, A. C.; Kowalski, R.; and Toni, F. 1992. Abductive Logic Programming. *Journal of Logic and Computation* 2(6):719–770.

Lifschitz, V. 1999. Answer set planning. In De Schreye, D., ed., *Logic Programming, Proc. 1999 Int. Conf. on Logic Programming (ICLP'99)*, 23–37. MIT Press.

Lifschitz, V. 2002. Answer Set programming and plan generation . *Artificial Intelligence* 138(1-2):39–54.

Marek, V., and Truszczyński, M. 1989. Stable semantics for logic programs and default reasoning. In E.Lust, and Overbeek, R., eds., *Proc. of the North American Conference on Logic Programming and Non-monotonic Reasoning*, 243–257.

Marek, V., and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In Apt, K.; Marek, V.; Truszczynski, M.; and Warren, D., eds., *The Logic Programming Paradigm: a 25 Years Perspective*. Springer-Verlag. pp. 375–398.

Moore, R. 1983. Semantical Considerations on nonmonotonic logic. In *Proc. of IJCAI-83*, 272–279.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25(3,4):241–273.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

van Emden, M., and Kowalski, R. 1976. The semantics of Predicate Logic as a Programming Language. *Journal of the ACM* 23(4):733–742.

Van Gelder, A.; Ross, K. A.; and Schlipf, J. 1991. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM* 38(3):620–650.

Woods, W. 1975. What's in a Link: Foundations for Semantic Networks. In Bobrow, D., and Collins, A., eds., *Representation and understanding: Studies in cognitive science*. Academic Press, New York. Also in Brachman and Levesque, *Readings in Knowledge Representation*, Morgan Kaufman, 1985.