About negation-as-failure and the informal semantics of Logic Programming

Marc Denecker[a,*], Miroslaw Truszczynski[b], Joost Vennekens[c]

[a]*Department of Computer Science*
KU Leuven
3001 Leuven, Belgium
[b]*Department of Computer Science*
University of Kentucky
Lexington, KY 40506-0633, USA
[c]*Department of Computer Science, KU Leuven*
Campus De Nayer
2860 Sint-Katelijne-Waver, Belgium

History and motivation. In its origin, logic programming was understood as the Horn fragment of first order logic (FO) augmented with a procedural interpretation induced by SLD inference (Kowalski, 1974). But already in 1975, the Prolog system developed in Marseille was extended with the *negation-as-failure* (NAF) inference rule, to derive negative literals `not` A (Roussel, 1975). The problem was that this inference rule is unsound with respect to the then accepted declarative interpretation of programs as Horn theories, as such theories do not entail negative literals $\neg A$. On the other hand, negation-as-failure had many useful applications in which the "unsound" conclusions were intuitively justified. This was the start of an extensive research program to study this phenomenon and to formalize it in a formal semantics of logic programming under which NAF inference would be sound. This led to the main formal semantics of LP that we know today by van Emden and Kowalski (1976), Reiter (1977), Clark (1978), Apt et al. (1988), Gelfond and Lifschitz (1988), and Van Gelder et al. (1991).

Despite this work, misunderstanding, controversies and disagreements remain about the informal semantics of logic programs and the meaning of its connectives. Two main and conflicting views on logic programming have emerged: the view of a logic program as a *definition*, and the view of a logic program as an *autoepistemic* theory. The former was proposed first by Keith Clark in his work on the completion semantics (Clark, 1978). After the Clark's completion semantics was criticized for its problems in the case of recursive programs (Harel, 1980), it was refined to the stratified semantics for the class of stratified programs (Apt et al., 1988), and to the well-founded semantics for arbitrary normal logic programs (Van Gelder et al., 1991). The autoepistemic view, resulting in the semantics of *stable models*, was presented first by Michael Gelfond and

*Corresponding author
Email addresses: `marc.denecker@cs.kuleuven.be` (Marc Denecker), `mirek@cs.uky.edu` (Miroslaw Truszczynski), `joost.vennekens@cs.kuleuven.be` (Joost Vennekens)

Vladimir Lifschitz (1988). It was developed on the basis of the autoepistemic logic by Moore (1984), who proposed it as a formal logic for expressing the knowledge of rational reflecting agents.

Ideas closely related to NAF inference appeared independently in the context of database query answering in Ray Reiter's work on the Closed World Assumption (Reiter, 1977). In 1980, Reiter generalized the Closed World Assumption in his default logic (Reiter, 1980), which he proposed as a formalism to study comonsense reasoning. Several years later, default logic inspired a novel semantics of logic programs proposed by Bidoit and Froidevaux (1987, 1991). This semantics was eventually shown to coincide with the stable model semantics (Marek and Truszczyński, 1989; Bidoit and Froidevaux, 1991). Since default logic is equivalent to the autoepistemic logic of Moore (Denecker et al., 2011), this line of research can be taken as an alternative embodiment of the autoepistemic view.

As for the informal semantics of the negation `not` in logic programs, many view it as a non-standard form of negation, different from classical negation. Gelfond and Lifschitz (1988) interpret `not` A as "*A is not known*" as a semantical way to reflect the fact that the NAF inference rule infers `not` A if *A is not provable*. While this is a compelling argument, it is easy to see that it is not a binding one. Assume that a logic has the property that its theories have *complete knowledge* on some class of formulas α (i.e., a theory T either entails α or entails $\neg\alpha$). If such logic possesses a sound and complete inference mechanism to infer $T \models \alpha$, then a (finite) failure of this engine to prove α ensures that $T \models \neg\alpha$. Thus, in all such cases, the negation-as-failure rule is a sound inference rule to infer *classically negated formulas*. Thus, to explain negation-as-failure, one can either interpret the symbol as an epistemic operator "I do not know that ...", or one can keep the standard objective interpretation of negation and explain logic programming as a logic that expresses complete knowledge.

As an aside, we note that in the early papers on the semantics of logic programming, including those by Clark (1978) and Apt et al. (1988), the term "negation-as-failure" refers exclusively to the non-standard negation-as-failure *inference rule* and not to the *negation connective*. In those papers, negation is interpreted as the *classical* negation. However, in the autoepistemic view of Gelfond and Lifschitz (1988) the negation cannot be treated as the classical one anymore. To stress that, researchers following the autoepistemic view started to use the term negation-as-failure also to refer to the `not` connective in programs.

Much of the controversy about the meaning of negation in logic programs stems from the coexistence of these two conflicting views. Perhaps one of the most confusing aspects herein is that at least to some extent, the *same* formal semantics can account for both views. Specifically, there is nothing in the mathematical definition of, say, the stable model semantics for logic programs that forces one to adopt either one view or the other. This means that in principle, two programmers may use the same program to compute the same output, and yet may have a completely different idea about what this program means! In order to properly address the question of negation-as-failure, it is therefore necessary to first discuss the informal semantics of a logic program.

Informal semantics. In declarative logics, formulas express information about the application domain. The role of the informal semantics theory of the logic is to specify

what this information is. Following Frege (1918), the information represented by a formula is a *thought*. To express this thought in a form accessible to others, we have to resort to natural language. And so, the informal semantics of a logic will be presented by a precisely defined translation of formulas into natural language. Let us illustrate this in the context of classical logic. For instance, consider the formula Φ:

$$\forall x : Human(x) \Leftrightarrow Man(x) \vee Woman(x)$$

The informal semantics of such an expression in the application domain depends on the meaning of its non-logical symbols in this application domain. We call this the *intended interpretation* $\mathcal{I}$ of the symbols and specify it as parametrized natural language sentences:

- $\mathcal{I}(Human)$ = "#1 is a human"
- $\mathcal{I}(Man)$ = "#1 is a man"
- $\mathcal{I}(Woman)$ = "#1 is a woman"

Given such an intended interpretation $\mathcal{I}$ for a vocabulary Σ of symbols, the translation to natural language providing the informal semantics $\mathcal{FO}_{\mathcal{I}}$ of FO expressions over Σ can now be defined inductively as specified in Table 1.

Φ	$\mathcal{FO}_{\mathcal{I}}(\Phi)$
x	x *(where x is a variable)*
$P(t_1, \ldots, t_n)$	$\mathcal{I}(P)\langle\mathcal{FO}_{\mathcal{I}}(t_1), \ldots, \mathcal{FO}_{\mathcal{I}}(t_n)\rangle$ *(i.e., the declarative sentence $\mathcal{I}(P)$ with its parameters instantiated to $\mathcal{FO}_{\mathcal{I}}(t_1), \ldots, \mathcal{FO}_{\mathcal{I}}(t_n)$)*
$\varphi \vee \psi$	$\mathcal{FO}_{\mathcal{I}}(\varphi)$ or $\mathcal{FO}_{\mathcal{I}}(\psi)$ (or both)
$\neg\varphi$	it is not the case that $\mathcal{FO}_{\mathcal{I}}(\varphi)$ *(i.e., $\mathcal{FO}_{\mathcal{I}}(\varphi)$ is false)*
$\varphi \Leftrightarrow \psi$	$\mathcal{FO}_{\mathcal{I}}(\varphi)$ if and only if $\mathcal{FO}_{\mathcal{I}}(\psi)$
$\forall x\ \varphi$	for all x in the universe of discourse, $\mathcal{FO}_{\mathcal{I}}(\varphi)$
$\ldots$	$\ldots$

Table 1: The informal semantics of FO formulas.

Aplying this definition to our formula Φ, the informal semantics $\mathcal{FO}_{\mathcal{I}}(\Phi)$ of Φ is given by the following sentence:

> For all x in the universe of discourse, x is a human if and only if x is a man or x is a woman (or both).

Assume that we take a different intended interpretation $\mathcal{I}'$, this time in the domain of mathematics, where $Human$ is interpreted as "$\#_1$ is an element of set A" and similarly

Man and $Woman$ stand for sets B and C, respectively. Now, the informal semantics $\mathcal{FO}_{\mathcal{I}'}(\varphi)$ of Φ is a mathematical statement, here given with the use of the standard notational shortcuts of mathematics:

> For all x in the universe of discourse, $x \in A$ if and only if $x \in B$ or $x \in C$ (or, equivalently, $A = B \cup C$).

For both application domains, the translation yields precise, extensional statements about their objective state of affairs. They are of the sort one finds in mathematical and scientific texts.

FO's theory of informal semantics also provides the interpretation of the formal semantical concepts such as the relation $\mathfrak{A} \models \varphi$ of satisfiability between a Σ-structure $\mathfrak{A}$ and a Σ-formula φ, where Σ stands for some fixed vocabulary. Namely, under an intended interpretation $\mathcal{I}$ of Σ, a Σ-structure is interpreted as an abstract representation of a state of affairs of the application domain. That is, an intended interpretation $\mathcal{I}$ specifies an abstraction function from states of affairs of the application domain to Σ-structures. Given a Σ-structure $\mathfrak{A}$ and a Σ-formula φ, $\mathfrak{A} \models \varphi$ informally means that $\mathcal{FO}_{\mathcal{I}}(\varphi)$ is *true* in the state of affairs abstracted by $\mathfrak{A}$ (under $\mathcal{I}$).

The theory of informal semantics of FO can be viewed as a precise falsifiable *hypothesis* about the forms of information (the "thoughts") expressible in FO. Indeed, the formal and informal semantics should match each other. That is, if a structure $\mathfrak{A}$ is an abstraction of a state of affairs of an application domain, then it must be that $\mathfrak{A} \models \varphi$ holds if and only if $\mathcal{FO}_{\mathcal{I}}(\varphi)$ is true in that state of affairs. This, of course, can never be proved since the informal semantics, despite its linguistic precision, is not mathematical. Nevertheless, the hypothesis can be subject to experiments and is in principle falsifiable. A possible experiment here could consist of a choice of a vocabulary Σ with an intended interpretation $\mathcal{I}$, a Σ-theory T and a Σ-structure $\mathfrak{A}$. The experiment would falsify the theory of informal semantics if $\mathfrak{A} \models T$ and $\mathcal{FO}_{\mathcal{I}}(T)$ would not be considered by human reasoners to be true in the states of affairs that abstract to $\mathfrak{A}$ under $\mathcal{I}$. Or, conversely, if it holds that $\mathfrak{A} \not\models T$ while $\mathcal{FO}_{\mathcal{I}}(T)$ would be considered by human reasoners to be true in all such states of affairs. That such mismatches have not occurred corroborates the theory of informal semantics of FO.

The definitional informal semantics of logic programming. We now review the informal semantics of logic programming cast in the style of the discussion above. We start with the *definitional* informal semantics corresponding to the view of programs as definitions. While there are no "official" linguistic rules on how to write an informal definition in a mathematical or scientific text, some conventions exist. Simple definitions often take the form of "if" or "if and only if" statements. More complex cases are inductive or recursive definitions, which are frequently represented as a set of informal rules, possibly with an induction order. When written according to these linguistic conventions, a definition has a precise and objective meaning to us. Consider an intended interpretation $\mathcal{I}$ for a vocabulary Σ, and a logic program Π in this vocabulary, with predicates $\{P_1, \ldots, P_n\}$. Assume also that $\Pi = \{A_1 \leftarrow \varphi_1, \ldots, A_m \leftarrow \varphi_m\}$, where φ_i is a conjunction of literals A and `not` A (with the conjunction connective represented by ','). Finally, assume that the free variables of the rule $A_i \leftarrow \varphi_i$, $1 \leq i \leq m$,

are $x^i_1, \ldots, x^i_{n_i}$. We define the following translation $\mathcal{OB}_{\mathcal{I}}(\Pi)$ of Π into natural language.

> *Relations $P^{\mathcal{I}}_1, \ldots, P^{\mathcal{I}}_n$ are the relations defined by the following (simultaneous) induction:*
>
> – *for all $x^1_1, \ldots, x^1_{n_1}$ in the universe, $\mathcal{FO}_{\mathcal{I}}(A_1)$ if $\mathcal{FO}_{\mathcal{I}}(\varphi_1)$*
> …
> – *for all $x^m_1, \ldots, x^1_{n_m}$ in the universe, $\mathcal{FO}_{\mathcal{I}}(A_m)$ if $\mathcal{FO}_{\mathcal{I}}(\varphi_m)$*

For completeness, to obtain a full account of the informal semantics of a logic program, an extra axiom is needed that expresses that the universe of discourse corresponds to the Herbrand universe. We will ignore this.

The proposition $\mathcal{OB}_{\mathcal{I}}(\Pi)$ is a statement that follows the linguistic conventions used to express inductive definitions. If Π is not recursive, the phrase "by the following simultaneous induction" should be dropped; what remains then is a definition by exhaustive enumeration, in which each rule represents one case. This translation makes use of the natural language connective "if" for the rule operator $\leftarrow$. When this word is used in the context of a case of an (inductive) definition, it is a *definitional* conditional. It has a precise and unambiguous meaning that differs from other conditionals such as material or strict implication. This is the "non-standard" connective in the language of logic programs under the definitional view. However, as implied be the use of $\mathcal{FO}_{\mathcal{I}}(\cdot)$ when interpreting the bodies of rules, `not` is interpreted as the standard objective negation, as in FO! Furthermore, nothing changes to the informal semantics of structures (abstractions of states of affairs) and to the satisfaction relation.

Thus, we now have a proposal for an informal semantics of logic programs. The question then is for what programs Π is $\mathcal{OB}_{\mathcal{I}}(\Pi)$ a sensible informal definition, and what formal semantics formalizes this interpretation. Inductive definability was the core concern of the development of the stratified semantics (Apt et al., 1988) and it is clear that (locally) stratified logic programs can be interpreted as inductive definitions. Expressing inductive definitions was also underlying the development of the well-founded semantics (Van Gelder et al., 1991). The links between this semantics and inductive definitions were recently investigated in detail by Denecker and Vennekens (2014). The punch-line is that logic programming under the well-founded semantics is a paraconsistent logic for expressing monotone inductive definitions, definitions by induction over a well-founded order, and the generalization of the two, called iterated inductive definitions. Logic programs with a two-valued well-founded model express mathematically correct definitions while programs with a three-valued well-founded model express a definition containing definitional paradoxes. In this view, negation-as-failure is actually classical negation. Below is an example.

Example 1. *The application domain is a simplified access control domain with a set of agents. One of them owns a file and has access to it. Any agent may delegate or block access to another agent. These delegations and blocks only take effect if the issuing agent has access. An agent has access if there is a path of delegations starting from the owner and ending with the agent such that no agent on the path is blocked. Consider the following program* Π.

$Delegates(A, B)$.
...
$Blocks(C, D)$.
...
$Access(Owner)$.
$Access(x) \leftarrow Access(y), Delegates(y, x), \texttt{not}\ EffectivelyBlocked(y)$.
$EffectivelyBlocked(x) \leftarrow Access(z), Blocks(z, x)$.

Under the obvious intended interpretation $\mathcal{I}$, the program expresses delegations and blocks by definition through exhaustive enumeration, and it expresses access and effective blocks through a mutually recursive definition over negation. The program is not locally stratified, not even when there are no $Block$ facts. The negation is as in FO, the standard objective negation. There is no central reflecting epistemic agent in this application domain.

Depending on the definitions of $Delegates$ and $Blocks$, the program illustrates several types of definitions and definitional paradoxes. If there are no $Blocks$ atoms, the definition collapses to the standard monotone inductive definition of reachability from $Owner$. If there is a hierarchy amongst agents with the owner on top such that agents delegate only to agents at the same or at a lower level of the hierachy, and block agents at lower levels, the definition is a fine iterated inductive definition*. Beyond that, there may or may not be definitional paradoxes. E.g., assume the owner gives access to agent A who blocks himself. In that case, the paradox arises that A has access exactly when he has no access. The well-founded model is three-valued on $Access(A)$ and $EffectivelyBlocked(A)$. Some definitional paradoxes, such as the definition of the truth predicate, have been under intensive scrutiny in philosophical logic.*

The autoepistemic informal semantics. We now recall the informal semantics of logic programs as autoepistemic theories, as it appeared in the papers by Gelfond and Lifschitz (1988; 1990), and which was formalized by the stable semantics. An alternative review of the informal semantics for this logic can be found in (Gelfond and Kahl, 2014, Section 2.2.1). For simplicity, here we only consider the propositional case. Logic programs with variables are interpreted by means of their so-called grounding that transforms them into propositional programs. We call this informal semantics of programs the autoepistemic informal semantics and we denote it by $\mathcal{GL}_{\mathcal{I}}$.

Table 2 shows the autoepistemic informal semantics $\mathcal{GL}_{\mathcal{I}}$ of extended programs, with negation-as-failure and classical negation. Again, we assume an intended interpretation $\mathcal{I}$ of the atoms of the program. As it is clear from this table, under $\mathcal{GL}_{\mathcal{I}}$, extended logic programs have both classical and non-classical connectives. On the one hand, the comma operator is the classical conjunction and the rule operator $\leftarrow$ is the classical material implication. Of the two negation operators, symbol $\neg$ is classical negation, whereas `not` is an epistemic negation, often called the *default negation*. The implicit composition operator (constructing the meaning of the program out of individual rules) is essentially standard conjunction, with this understanding that the agent knows *only* what is explicitly stated. Nonmonotonicity arises due to the introspective nature of the negation by default: adding a rule may lead to a new belief A, thus falsifying `not` A, which may turn previously believed literals into unbelieved.

Φ	$\mathcal{GL}_{\mathcal{I}}(\Phi)$
propositional atom A	$\mathcal{I}(A)$
propositional literal $\neg A$	it is not the case that $\mathcal{I}(A)$
expression of the form `not` C	the agent does not know that $\mathcal{GL}_{\mathcal{I}}(C)$
expression of the form Φ_1, Φ_2	$\mathcal{GL}_{\mathcal{I}}(\Phi_1)$ and $\mathcal{GL}_{\mathcal{I}}(\Phi_2)$
rule $Head \leftarrow Body$	if $\mathcal{GL}_{\mathcal{I}}(Body)$ then $\mathcal{GL}_{\mathcal{I}}(Head)$ *(in the sense of material implication)*
program $P = \{r_1, \ldots, r_n\}$	All the agent knows is: • $\mathcal{GL}_{\mathcal{I}}(r_1)$ and • ... • $\mathcal{GL}_{\mathcal{I}}(r_n)$

Table 2: The Gelfond-Lifschitz (1988; 1990) autoepistemic informal semantics for ASP formulas.

On the formal level, the key semantic structure is a set X of FO-literals, an *answer set*. Mathematically, if that set contains only atoms, it has the same format as a structure of propositional logic. However, its informal semantics is completely different. In $\mathcal{GL}_{\mathcal{I}}$, a (consistent) set X of literals is viewed as an abstraction of a *belief state* of some agent. An agent in a belief state considers certain states of affairs as possible and the others as impossible. What these possible states of affairs are is not represented in this semantics. Instead, the state of belief is abstractly represented by the set X of believed literals, those that are true in all the possible states of affairs. Specifically, X is the set of literals such that $\mathcal{I}(L)$ is true in all states of affairs considered possible in the state of belief. A negation-by-default literal `not` L holds true in X if $L \notin X$, that is, if L is false in at least one possible state of affairs. Thus, the informal semantics $\mathcal{GL}_{\mathcal{I}}$ explains the meaning of programs in terms of what literals an agent with incomplete knowledge of the application domain might believe in.

It was argued by Gelfond and Lifschitz (1988, 1990) (using a mapping to autoepistemic logic) that the answer sets of an extended logic program Π correspond one to one to the potential states of belief of an agent holding $\mathcal{GL}_{\mathcal{I}}(\Pi)$ to be true. In other words, given that $\mathcal{GL}_{\mathcal{I}}(\Pi)$ represents precisely the knowledge of the agent, X could be the set of literals the agent believes.

If Π is a logic program (without classical negation), the state of affairs in which all atoms are true satisfies all rules since they make their heads true. It follows that no negative literal is believed. Hence, all stable models are sets of atoms only. As said before, such sets mathematically correspond to structures of propositional vocabularies. However, it is a mistake to interpret such sets as abstractions of possible states of affairs.

To illustrate this informal semantics, let us consider the well-known *interview* example of Gelfond and Lifschitz (1991).

Example 2. *Whether students of a certain school are eligible for a scholarship de-*

pends on their GPA and on their minority status. The school has an incomplete database about candidate students. Students for which the school has insufficient information to decide eligibility should be invited for an interview. The following ELP program expresses the school's knowledge.

$$
\begin{aligned}
Eligible(x) &\leftarrow HighGPA(x).\\
Eligible(x) &\leftarrow FairGPA(x), Minority(x).\\
\neg Eligible(x) &\leftarrow \neg FairGPA(x), \neg HighGPA(x).\\
Interview(x) &\leftarrow \texttt{not}\ Eligible(x), \texttt{not}\ \neg Eligible(x)\\
Minority(brit).&\\
HighGPA(mary).&\\
\neg Minority(david).&\\
FairGPA(david).&
\end{aligned}
$$

The three rules for $Eligible$ specify a partial *policy for eligibility: they determine the eligibility for all students except for non-minority students with fair GPA. In this sense, under $\mathcal{GL}_{\mathcal{I}}$, this program does not actually* define *when a student is eligible. The next rule is* epistemic. *It expresses that the school interviews a person whose eligibility is unknown. The remaining rules specify partial data on students Mary, Brit and David. In particular, $FairGPA(brit)$ and even $FairGPA(mary)$ are unknown. This is all the school knows.*

*For Mary, the first rule applies and the school knows that she is eligible. The epistemic fourth rule will therefore not conclude that she should be interviewed. Incidentally, nor is it implied that she will not be interviewed; the school (formally, the program) simply does not know. This follows from the informal semantics of the implicit composition operator: "*all *the agent knows is. . . ". For Brit and David, their eligibility is unknown. However, the reasons are different: for Brit because of lack of data, for David because the policy does not specify it. Therefore, both will be interviewed. The unique answer set extends the student data with the following literals:*

$$Eligible(mary), Interview(brit), Interview(david).$$

The crucial property of this example is that it is inherently epistemic. Whether a student should be interviewed does not only depend on the properties of the student in the objective state of affairs, but also on the schools knowledge about this student. Because of this property, an epistemic logic is required here.

The interview example, with its inherent reflexive component, is a clear case where the informal semantics $\mathcal{GL}_{\mathcal{I}}$ applies. It is written in the language of extended logic programming, for which the definitional informal semantics is not defined. For a comparison of the two informal semantics on the *same* program, consider the next example.

Example 3. *Consider the following propositional program Π :*

$$\{\ P \leftarrow \texttt{not}\ Q\ \}$$

Its unique stable, well-founded or completion model is $\{P\}$. For a first intended interpretation $\mathcal{I}_1$, the situation is that a man gets separated from his family after an earthquake and has no information about their safety. Let $\mathcal{I}_1(P)$ mean that the man is unhappy and $\mathcal{I}_1(Q)$ that his family is safe. Under $\mathcal{GL}_{\mathcal{I}_1}$, the program expresses that all the man knows is that if he does not know that his family is safe, then he is unhappy. The stable model abstracts the unique possible state of belief wherein he does not know whether his family is safe and he is unhappy. Under the definitional informal semantics $\mathcal{OB}_{\mathcal{I}_1}$, the program contains the empty definition for Q which expresses that his family is not safe; furthermore it expresses that the man is unhappy if and only if his family is not safe (objective negation). Under this reading, the program expresses complete knowledge admitting a unique possible state of affairs where the family is not safe and the man is unhappy. To illustrate the difference between the two interpretations, we consider the state of affairs abstracted as the structure $\{P, Q\}$ where the man is unhappy and his family is safe (but the man is not aware of this). This is a possible state of affairs in the autoepistemic informal semantics and impossible in the definitional semantics. Here clearly the autoepistemic reading is correct while the definitional one is incorrect.

For an alternative intended interpretation $\mathcal{I}_2$ where the definitional informal semantics is correct and the autoepistemic one is not, let $\mathcal{I}_2(Q)$ mean that Lincoln is alive and $\mathcal{I}_2(P)$ that he is dead. It is clear that under this intended interpretation, the definitional, objective informal semantics gives a reading of this program that is true in the world (Lincoln is not alive, and he is dead if and only if he is not alive) while the autoepistemic one does not. The state of affairs abstracted as $\{P, Q\}$ where Lincoln is both alive and dead is impossible in this scenario.

Discussion. The problem of negation-as-failure as it arose in the 1970s was in the first place a problem with the informal semantics for logic programming. Over the years, two informal semantics for LP were proposed: the autoepistemic one $\mathcal{GL}_{\mathcal{I}}$ and the definitional one $\mathcal{OB}_{\mathcal{I}}$. They assign different meaning to the main connectives $\leftarrow$ and `not`. In one case, negation is non-standard epistemic, in the other it is standard classical objective negation. They also assign different meaning to the semantic concepts (interpretations, answer sets, models, $\models$) making it possible that the same program under the same formal semantics can be explained differently in the two informal semantics.

Both informal semantics make sense. For both, knowledge scenarios in application domains are found that are naturally expressed by logic programs under the specific informal semantics. However, they are fundamentally different. When under some intended interpretation $\mathcal{I}$, one of the informal semantics gives a reading of a program Π that is true in the domain, the other does not and vice versa.

While both informal semantics make sense, there is an important qualitative difference between them. The definitional informal semantics explains the meaning of logic programs in terms of precise, objective definitions of the kind that are used in mathematical and scientific text. Objective definitional knowledge is available in great quantities in many applications. On the other hand, autoepistemic reasoning was developed as a principle for commonsense reasoning. It is a complex form of knowledge and it is subject to many different interpretations (Denecker et al., 2011). As such, the autoepistemic reading of programs is often hard to understand.

Independent of the informal view one takes, pure logic programming poses serious limitations as a knowledge representation language, and needs to be extended. However, what these limitations are, and hence, in what direction the language should be extended, depends strongly on the informal semantics one works with. In the definition-based informal semantics, the problem is that no incomplete knowledge can be represented since all predicates need to be defined. The natural extension therefore is to drop LP's assumption that a program should define *every* predicate and to consider definitions that define one or more symbols in terms of other parameter symbols. This road was been followed in the logic FO(.), in which such generalized definitions were integrated in classical logic (Denecker, 2000; Denecker and Ternovska, 2008) to compensate the latter's weakness on inductive definability.

In the autoepistemic interpretation, LP's weakness is that no negative information can be expressed which, as explained above, led Gelfond and Lifschitz to extend LP with classical negation resulting in ELP. Even then, ELP is not an expressive KR language for expressing autoepistemic knowledge compared to full autoepistemic logic (Moore, 1984, 1985) and default logic (Reiter, 1980).

Recently, the language of ELP has flourished as a method for representing and solving computational search problems in the domain of Answer Set Programming. Crucial in this development has been the emergence of the Generate-Define-Test (GDT) paradigm for constructing ASP programs. Over time, to support GDT programming, answer set programming has developed a richer language with new features including choice rules, aggregates and weight constraints. Interestingly, the applications to which this GDT methodology is typically applied involve only *objective* knowledge, and they lack an introspective epistemic agent. The Gelfond-Lifschitz informal semantics is therefore not suitable for these GDT programs. The new language extensions so far eluded explanation in the autoepistemic view.

In (Denecker et al., 2012) we showed that such programs can be split into theories similar as those of the logic FO(.): combinations of, essentially, objective FO axioms and definitions. For such programs, the natural interpretation of answer sets is as abstractions of possible states of affairs, and correspondingly, the informal semantics of `not` is standard classical negation.

Finally, we hope to have showed in this paper that analysis of the informal semantics of LP and ASP is fruitful for understanding their declarative foundations and for their use and further development as KR languages.

References

Apt, K. R., Blair, H. A., Walker, A., 1988. Towards a theory of declarative knowledge. In: Minker, J. (Ed.), Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, pp. 89–148.

Bidoit, N., Froidevaux, C., 1987. Minimalism subsumes default logic and circumscription. In: Proceedings of IEEE Symposium on Logic in Computer Science, LICS-87. IEEE Press, pp. 89–97.

Bidoit, N., Froidevaux, C., 1991. Negation by default and unstratifiable logic programs. Theoretical Computer Science 78 (1, (Part B)), 85–112.

Clark, K. L., 1978. Negation as failure. In: Logic and Data Bases. Plenum Press, pp. 293–322.

Denecker, M., 2000. Extending classical logic with inductive definitions. In: Lloyd, J. W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Palamidessi, C., Pereira, L. M., Sagiv, Y., Stuckey, P. J. (Eds.), CL. Vol. 1861 of LNCS. Springer, pp. 703–717.

Denecker, M., Lierler, Y., Truszczyński, M., Vennekens, J., 2012. A Tarskian informal semantics for answer set programming. In: Dovier, A., Costa, V. S. (Eds.), ICLP (Technical Communications). Vol. 17 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 277–289.

Denecker, M., Marek, V., Truszczyński, M., 2011. Reiter's default logic is a logic of autoepistemic reasoning and a good one, too. In: Brewka, G., Marek, V., Truszczyński, M. (Eds.), Nonmonotonic Reasoning – Essays Celebrating Its 30th Anniversary. College Publications, pp. 111–144.
URL http://arxiv.org/abs/1108.3278

Denecker, M., Ternovska, E., Apr. 2008. A logic of nonmonotone inductive definitions. ACM Trans. Comput. Log. 9 (2), 14:1–14:52.
URL http://dx.doi.org/10.1145/1342991.1342998

Denecker, M., Vennekens, J., 2014. The well-founded semantics is the principle of inductive definition, revisited. In: Baral, C., De Giacomo, G., Eiter, T. (Eds.), KR. AAAI Press, pp. 1–10.
URL http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7957

Frege, G., 1918. Beitrge zur Philosophie des deutschen Idealismus I. 2. Ch. Der Gedanke: eine logische Untersuchung.

Gelfond, M., Kahl, Y., 2014. Knowledge Representation, Reasoning, and the Design of Intelligent Agents. Cambridge University Press, Cambridge, UK.

Gelfond, M., Lifschitz, V., 1988. The stable model semantics for logic programming. In: Kowalski, R. A., Bowen, K. A. (Eds.), ICLP/SLP. MIT Press, pp. 1070–1080.
URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.6050

Gelfond, M., Lifschitz, V., 1990. Logic programs with classical negation. In: Warren, D. H. D., Szeredi, P. (Eds.), ICLP. MIT Press, pp. 579–597.

Gelfond, M., Lifschitz, V., 1991. Classical negation in logic programs and disjunctive databases. New Generation Computing 9 (3/4), 365–386.

Harel, D., 1980. Review on logic and data bases.

Kowalski, R. A., 1974. Predicate logic as programming language. In: IFIP Congress. pp. 569–574.

Marek, W., Truszczyński, M., 1989. Stable semantics for logic programs and default theories. In: E.Lusk, Overbeek, R. (Eds.), Proceedings of the North American Conference on Logic Programming. MIT Press, pp. 243–256.

Moore, R. C., 1984. Possible-world semantics for autoepistemic logic. In: Proceedings of the Workshop on Non-Monotonic Reasoning. pp. 344–354, reprinted in: M. Ginsberg, ed., *Readings on Nonmonotonic Reasoning*, pages 137–142, Morgan Kaufmann, 1990.
URL http://www.sri.com/sites/default/files/uploads/publications/pdf/616.pdf

Moore, R. C., 1985. Semantical considerations on nonmonotonic logic. Artif. Intell. 25 (1), 75–94.
URL http://dx.doi.org/10.1016/0004-3702(85)90042-6

Reiter, R., 1977. On closed world data bases. In: Gallaire, H., Minker, J. (Eds.), Logic and Data Bases. Plenum Press, pp. 55–76.

Reiter, R., 1980. A logic for default reasoning. Artif. Intell. 13 (1-2), 81–132.
URL http://dx.doi.org/10.1016/0004-3702(80)90014-4

Roussel, P., 1975. Prolog, manuel de référence et d'utilisation. Tech. rep., Groupe Intelligence Artificielle, Faculté des Science de Luminy, Université Aix-Marseille II, France, September.

van Emden, M. H., Kowalski, R. A., 1976. The semantics of predicate logic as a programming language. J. ACM 23 (4), 733–742.
URL http://dx.doi.org/10.1145/321978.321991

Van Gelder, A., Ross, K. A., Schlipf, J. S., 1991. The well-founded semantics for general logic programs. J. ACM 38 (3), 620–650.
URL http://dx.doi.org/10.1145/116825.116838