Interpretations of Negations in Logic Programs

Catholijn M. Jonker

Keywords: Negations, Belief operator, Autoepistemic Logic.

Abstract

We consider logic programming languages in which it is possible to derive negative information both through a direct and through an indirect derivation mechanism. The isolated direct derivation steps are monotone and the indirect are based on the Closed World Assumption and therefore nonmonotone. The semantics of the direct and indirect negative conclusions have separately been studied extensively, as can be seen from the work on Definite Logic Programs with classical negation and the semantics of Normal Logic Programming.

However, by combining both direct and indirect inferences of negative conclusions in one system, the semantics can no longer be monotone, because of the nonmonotone derivation mechanism of the indirect derivations. On the other hand, it can no longer be the same nonmonotone derivation mechanism that corresponds to the indirect derivations, since there are additional mechanisms to derive falsity. We compare two approaches, one in which two different interpretations of negative conclusions are considered, the other in which negative conclusions are interpreted uniformly.

1 Introduction

Knowledge representation languages that are to be of practical use require at least options for drawing negative conclusions. It must be possible to derive such information via direct means, but for practical reasons it must also be possible to derive negative information by the absence of other information. In other words, it must be possible to prove negative statements and it must be possible to infer a negative statement since the corresponding positive statement is believed to be unprovable. One of the application areas in which both ways to use negations commonly occur is the area of knowledge-based systems. For example, in implementation environments for the development of expert systems, like AionDS [26] and Nexpert Object [14], negation is commonly used not only in the body of rules, but also in the conclusions. In conclusions negation is allowed to appear in the form of negative truth assignments to boolean variables. Next to using rules with such negative conclusions it is also possible to derive negative information by the application of the Closed World Assumption. Also in Desire [11], a specification language for compositional knowledge-based system, rules can have negative conclusions and negative statements in the bodies of rules. In principle the negation is interpretated as a classical negation both in the body and in the conclusion of the rule. In addition the Closed World Assumption can explicitly be expressed by meta-knowledge in order to nonmonotonically derive negative information.

Although frequently used in various applications, it is not clear what the semantics of these negative conclusions is. If the requirements for direct and indirect derivation of negative conclusions are considered separately, the issue is clear. The first corresponds to a monotone derivation mechanism for falsity and the second to a nonmonotone mechanism. However, by combining both requirements in one system, the semantics can no longer be monotone, because of the nonmonotone derivation mechanism of the second requirement. On the other hand, it can no longer be the same nonmonotone derivation mechanism that corresponds to the second requirement, since there are additional mechanisms to derive falsity. One approach is to continue considering the two different

interpretations, i.e., negative conclusions derived in a direct way are interpreted as *proven to be false*, while the indirect negative conclusions are interpreted as *believed to be unprovable*. The other approach is to look for one uniform interpretation. These two approaches can easily be formalised as extensions of Normal Logic Programming. For the first approach a different negation symbol is associated with each of the interpretations. For the second approach one negation symbol suffices. One can then compare the approaches by embedding both extensions of Normal Logic Programming into one well-understood nonmonotonic formalism.

There exist several extensions of logic programming in which both ways to draw negative conclusions are possible. We mention the work of Gelfond and Lifschitz [4], Jonker [6, 7], Minker [13], Pereira et al. [1, 17, 19] and Przymusinski [21, 24, 25]. Their work resulted in extensions of the well-known anser-set, stable, stationary, supported and wellfounded semantics for the respective extensions of logic programming.

In this paper two of these formalisms are studied; an extension with two negation symbols and an extension with only one negation symbol. For the extension with two negation symbols we choose Extended Logic Programming, for the other we choose Imex Logic Programming.

Extended Logic Programming extends Normal Logic Programming with an extra negation symbol, called *explicit* negation. The explicit negation symbol corresponds to the *direct* inference of negative conclusions. The other negation symbol is called *implicit* negation, it corresponds to the *indirect* inference of negative information. The explicit negation is allowed to appear both in the conclusions and in the bodies of rules. The two negations are connected through the so called *Coherence Principle* of Pereira and Alferes [17]. This principle expresses that if a statement A is explicitly false, then it is also implicitly false (but not necessarily the other way round). In other words, if a negative conclusion can be derived by direct means, it is also indirectly derivable. Furthermore, if a statement is true, then its explicit negation must be implicitly false. In symbols:

$$S \quad \text{implies} \quad not\ \overline{S} \tag{1}$$

where S is either an atomic symbol or an explicitly negated symbol, $\overline{S}$ is the complement of S with respect to the explicit negation, and "*not*" is the symbol for implicit negation. For the syntax and semantics we refer to Section 2, for more information and definitions of various declarative semantics we refer to [4, 10, 18, 16, 22, 23, 27, 28] and [29].

Imex Logic Programming is the extension of Normal Logic Programming in which the existing negation is allowed to appear both in the conclusions and in the bodies of rules. For the syntax and semantics we refer to Section 2, for more information and definitions of various declarative semantics we refer to [6, 7, 8, 9].

For the well-understood nonmonotonic formalism with which Extended and Imex Logic Programming will be compared we choose Przymusinski's Autoepistemic Logic with Minimal Beliefs (AELB). AELB is a non-monotonic knowledge representation framework that augments Moore's autoepistemic logic with an additional minimal belief operator $\mathcal{B}$. AELB has turned out to be very successful for the comparison of logic programming languages. In [24] and [25] Przymusinski reports AELB as a uniform semantical framework that isomorphically contains the various semantics for Normal, Extended, and Disjunctive Logic Programming. For an overview of the existing relationships between Logic Programming and AELB or several other nonmonotonic formalisms we refer to [2].

In Section 2 the formal syntax and an informal interpretation of Extended and Imex Logic Programming are given.

Section 3 contains a recapitulation of Przymusinski's Autoepistemic Logic of Minimal Belief.

In Section 4 we discuss the existing informal interpretations of the negations of Extended and Imex Logic Programming.

In Section 5 the negations are formally interpreted by the embedding of Extended and Imex Logic Programming into AELB. It contains the theorems that relate the declarative semantics of the Logic Programs to the corresponding AELB-theories, and the theorems that relate Extended to Imex Logic Programming.

In Section 6 the results of this paper are discussed.

2 Logic Programs

The first researchers to propose forms of Logic Programming with some kind of explicit or strong negation symbol next to an implicit negation symbol are Gelfond and Lifschitz [4], Pearce and Wagner [15], Przymusinski (see for example [23]), Pereira [17] and Jonker [6]. A lot of work has been done on the declarative semantics of such programs, we mention the work of Gelfond and Lifschitz [4], Jonker [6, 7, 8, 9], Minker [13], Pereira et al. [17, 19, 1] and Przymusinski [21, 24, 25]. Their work resulted in extensions of the well-known anser-set, stable, stationary, supported and wellfounded semantics.

In this paper the alphabet of language $\mathcal{K}_E$ consists of the atomic symbols A, A_0, A_1, ... of $\mathcal{K}_B$, the negation symbols "$-$", for explicit negation, "*not*" for implicit negation, and the symbols "$\leftarrow$" and ",". The informal interpretation of the implicit negation "*not*", that is under discussion in this paper, is *believed to be false*. The symbol "$\leftarrow$" is a unidirectional logical symbol that informally means that if the right-hand side is true, then so is the left-hand side. The symbol "," is to be read conjunctively. The informal interpretation of the explicit negation symbol "$-$", that is under discussion in this paper, is *proved to be false*. For a formal definition of an interpretation of $\mathcal{K}_E$ we refer the reader to the Appendix. The literals of $\mathcal{K}_E$ are all expressions of the form A, $notA$, $-A$ and $not{-}A$, where A is an atom. An Extended Logic Programming rule is an expression of the form:

$$S_0 \quad \leftarrow \quad S_1, \ldots, S_m, notS_{m+1}, \ldots, notS_n$$

where each S_i $(0 \leq i \leq n)$ is either an atom A or an explicitly negated atom $-A$. An Extended Logic Program consists of a finite set of Extended Logic Programming rules. We define ELP to be the class of all Extended Logic Programs.

The two negations are connected through the *Coherence Principle* of Pereira and Alferes [17], see Equation 1.

Example 1 Consider the following Extended Logic Program:

$$\begin{aligned} A &\leftarrow not\ B \\ B &\leftarrow not\ A \\ -A &\leftarrow C \\ C &\leftarrow not\ D \end{aligned}$$

The first two rules form an odd loop well-known in Normal Logic Programming. Based on these two rules only it is impossible to decide whether A or B should be false and thus whether B or A should be true. (Therefore, these two rules would have two stable models, one for each of the possibilities.) The implicit negative statement $notD$ will be true, since there is no rule with D as its head. As a consequence, C is true. Since C is true, it is possible to explicitly derive $-A$. By the Coherence Principle, $notA$ will be true. Finally, B is true, since $notA$ is true. ■

Imex Logic Programming has been developed as an extension of Normal Logic Programming in which the only negation symbol is allowed to appear both on the left- and on the right-hand side [6]. The imex negation symbol has the properties of the implicit negation of Normal Logic Programming and some additional explicit properties, hence the name Imex negation.

The alphabet of $\mathcal{K}_I$ is the same as that of $\mathcal{K}_E$, except for the negation symbols. $\mathcal{K}_I$ has only one negation symbol: "$\sim$". The informal interpretation of "$\sim$", that is under discussion in this paper, is *believed to be false*. For a formal interpretation of the language $\mathcal{K}_I$ we refer the reader to the Appendix. The literals of $\mathcal{K}_I$ are all expressions of the form A and $\sim A$, where A is an atom. An Imex Logic Programming rule is an expression of the form:

$$S \leftarrow A_1, \ldots, A_m, \sim A_{m+1}, \ldots, \sim A_n$$

where each A_i $(1 \leq i \leq n)$ is an atom A and S is either an atom A or an imex negated atom $\sim A$. An Imex Logic Program is a finite set of Imex Logic Programming rules. We define IMEX to be the class of all Imex Logic Programs.

Example 2 Consider the following Imex Logic Program:

$$\begin{aligned} A &\leftarrow \sim B \\ B &\leftarrow \sim A \\ \sim A &\leftarrow C \\ C &\leftarrow \sim D \end{aligned}$$

The informal reasoning according to this Imex program is similar to that of Example 1, with the exception of the use of the Coherence Principle. Since imex negation has all properties of implicit negation $\sim D$ will be true, since there is no rule with D as its head. As a consequence, C is true. Using the truth of C, it is possible to explicitly derive $\sim A$. In other words $\sim A$ will be true. Finally, B is true, since $\sim A$ is true. ■

In this paper a Logic Program is either an Extended or an Imex Logic Program. The literal on the left-hand side of a program rule r is called the *head* (written $hd(r)$) and the sequence on the right-hand side is called the *body* of the rule. We also allow countably many variables and function symbols in the languages $\mathcal{K}_I$ and $\mathcal{K}_E$. However, in the interpretations as defined in the Appendix, we only consider ground formulas. The *Herbrand base* $\mathcal{H}_P$ of Logic Program P is the collection of all ground instances of atoms that occur somewhere in P.

3 Autoepistemic Logic of Minimal Belief

The language of AELB, first published in [21], is a propositional modal language $\mathcal{K}_B$ with standard connectives $(\wedge, \vee, \supset, \neg)$ and a modal operator $\mathcal{B}$. The operator $\mathcal{B}$ represents *belief*. The formulas in which $\mathcal{B}$ does not occur are called *objective* formulas. We define the language $\mathcal{K}$ to be the language $\mathcal{K}_B$ but without the modal operator. Any theory T in the language $\mathcal{K}_B$ is called an *autoepistemic* theory. The intended meaning of $\mathcal{B}F$ is "F is believed", or, more precisely, "F can be non-monotonically inferred", i.e., $T \models_{nm} F$, where $\models_{nm}$ denotes a fixed non-monotonic inference relation. In general, different non-monotonic inference relations can be used. In [24] Przymusinski used *circumscription* ([12]), represented by $\models_{min}$. As axioms he assumed, for any $\mathcal{K}_B$-formulas F and G, the *conjunctive belief axiom*:

$$\mathcal{B}(F \wedge G) \equiv \mathcal{B}F \wedge \mathcal{B}G \tag{2}$$

and the *consistency axiom*:

$$\mathcal{B}F \supset \neg\mathcal{B}\neg F \tag{3}$$

Definition 3 *[24] The set of* consequences $Cn^*(T)$ *of a given theory* T *is defined as the set of logical consequences, denoted by* Cn*, of the theory* T *augmented with the two axioms (2) and (3):*

$$Cn^*(T) = Cn(T \cup \{(2), (3)\}).$$

A derivability *relation* $\vdash^*$ *is defined by stating that a formula* F *is derivable from* T *iff it belongs to* T*'s consequences:* $T \vdash^* F$ *iff* $F \in Cn^*(T)$*.*

Definition 4 *A theory* T^* *is a* static expansion *of* T *iff it satisfies the following fixpoint equation:*

$$T^* \quad = \quad Cn^*(T \cup \{\mathcal{B}F : T^* \models_{min} F\}),$$

where F *ranges over all formulas of* $\mathcal{K}_B$*.*

We further define

- T is *consistent* if $T \nvdash^* \bot$.
- The set of all static expansions of T is denoted by $E(T)$. Note that if $T_1 \vdash^* Cn^*(T_2)$ and vice versa for two theories T_1 and T_2, then $E(T_1) = E(T_2)$.
- $Cons_3(T) = \{S \in E(T) \mid S \text{ is consistent } \}$.
- $Cons_2(T) = \{S \in Cons_3(T) \mid \text{for all objective atoms } A \text{ either } \mathcal{B}\neg A \in S \text{ or } \mathcal{B}A \in S\}$.
- The least static expansion T^C of T is called the *static completion* of T.
- The *static semantics* $Stat(T)$ of T is the set of all formulas that belong to the static completion T^C of T.
- $\mathcal{K}_B$ theories T_1 and T_2 are *equivalent* iff $E(T_1) = E(T_2)$.

Note that according to Definition 4 $T \vdash^* F$ also implies that $T \vdash^* \mathcal{B}F$.

4 Interpreting negations

Przymusinski showed how Circumscription, Moore's autoepistemic logic, the stable, the stationary, and the wellfounded semantics of (Disjunctive) Logic Programs can be given an formal interpretation by embedding them into AELB. Przymusinski also showed that the stable semantics of Extended Logic Programs can be embedded into AELB using an embedding operator, denoted in this paper by K_{ELP}. Alferes and Pereira [2] continued this work on Extended Logic Programming and showed that K_{ELP} also embeds the stationary and the wellfounded semantics of such programs into AELB. By K_{ELP}-embedding a semantics of Logic Programming into AELB we mean that the models of the semantics with respect to a program P correspond in a one-to-one way to a specific set of expansions of the AELB-theory $K_{ELP}(P)$. For example, the stable semantics of Extended Logic Programming are K_{ELP}-embeddable into AELB, since the stable models of an Extended Logic Program P correspond one-to-one with those consistent expansions of the AELB-theory $K_{ELP}(P)$ that contain for each atomic symbol A of P either $\mathcal{B}A$ or $\mathcal{B}\neg A$.

One of the conclusions drawn in [2] is that the embedding of Extended Logic Programming into AELB with respect to the stable, stationary and wellfounded models shows that explicit negation stands for "proving falsity" whereas the meaning of the implicit negation is "believed to be not proven". Indeed, these remarks correspond to the initial intention of having two different kinds of negations for the formalisation of the use of negations in practical knowledge representation. On the other hand, it is not clear what the semantics of these negative conclusions is. If the requirements for direct and indirect derivation of negative conclusions are considered separately, the interpretations as given in [2] work very well indeed. However, by combining both requirements in one system as is done in commonly used knowledge-representation languages, the semantics of the

directly derived negative conclusions can no longer be monotone, because of it might have been derived using the implicit negation which is interpreted in terms of belief.

Two small examples demonstrate this clearly in Extended Logic Programming. For the informal interpretation of the negation symbols we take the reading of [2], that is "$-A$" means "A is provably false" and "$notB$" means "it is believed that B is not proven". In other words, the following Extended Logic Programming rule

$$-A \leftarrow notB$$

can be read as: if it is believed that B is not proven, then A is proven false. We find this reading unsatisfactory, since in this interpretation the provability of falsehood of a statement can depend on the believed unproveness of an other statement or even on the belief that the same statement is unproven (take A and B the same). One would expect that conclusions on the basis of only a belief are themselves only beliefs. At least it is clear that the notion *proven* and the notion *belief* are closely related in the sense that a belief can be used as a proof and a proof is a good reason for belief. Therefore, it would be better if the $-A$ is also interpreted in terms of belief. Why not as "believe that A is false"? If we would continue this line of reasoning we might separate proofs that do not make use of implicit negation (monotonic inferences) from those that do (nonmonotonic inferences). In particular, we could also make the same distinction for the positive literals of the program, see [7], Section 5.3. It would be worth while to investigate how this relates to the notion of (maximal) deductive base of a nonmonotonic inference operation as discussed in [5]. However, in this paper we will restrict ourselves to considering the interpretation of the negations.

The second example concerns the Coherence Principle, see Equation 1. The Coherence Example expresses that if $-A$ is true, then $notA$ must be true as well. In other words, if the falsity of A is provable, then A itself is believed to be unproven. This interpretation is not satisfactory in the following example.

Example 5

$$\begin{aligned} -A &\leftarrow \\ A &\leftarrow notB \end{aligned}$$

Interpreting this example, we see that A is provably false (since $-A$ is unconditionally true). Furthermore, $notB$ must be true, since there is no rule with B on the left-hand side that could prove B. Therefore, A is true as well. Since $-A$ is true, $notA$ must be true according to the Coherence Principle. In other words, it is believed that A is unproven, whereas a proof for A exists in the form of $notB$.

Even if provable is interpreted as nonmonotonically provable, then the problem is still there. Because then the notions belief and provable coincide.

Of course, this problem can be circumvented by dropping the Coherence Principle. However, in [17], Pereira and Alferes give a highly convincing argumentation for the necessity of the Coherence Principle. Basically, without the Coherence Principle, Extended Logic Programming does not really extend Normal Logic Programming, since every $-A$ can then be interpreted as a new atomic symbol $\overline{A}$, see [7, 17, 20].

Since we rather do not want to change the interpretation of the implicit negation, we seek a solution to this problem in a slightly different interpretation of the explicit negation symbol, namely *believed to be false* instead of *proving falsity*. As can be noticed by this choice both negations have the same interpretation. Therefore, they can be denoted by one symbol. This is in agreement with common practice in knowledge-representation languages as employed in environments for the

development of knowledge-based systems. The informal interpretation of imex negation is the same as that of the implicit negation. In practice the belief that a statement A is false, or equivalently that a statement $\sim A$ is true, can be based on impossibility to derive A, but also on the derivability of $\sim A$ from other beliefs.

The property imex negation shares with explicit negation is that it is possible to make explicitly negative statements, e.g.,

$$\sim A \quad \leftarrow$$

In order to find an alternative interpretation of the explicit negation symbol we embed Imex Logic Programming into AELB using a mapping denoted by K_{IMEX}. Here, K_{IMEX} is an adaptation for Imex Logic Programming of the K-translation, in this sense it is a continuation of the work of Przymusinski, Alferes and Pereira. The basic ingredient of the translation of Imex Logic Programs is the translation of an Imex negated statement $\sim A$ into $\mathcal{B}\neg A$.

5 Programs as AELB-theories

A logical language can semantically be interpreted by a mapping to a well-understood logic. Such a mapping enables a deeper analysis of the semantics of such a logical language. If such a mapping is made for two logical languages into the same well-understood logic, then the two logical languages can be compared by comparing their images under the mapping. One of the interesting elements in such a comparison is a notion of equivalence. In this paper, we map formulas (and thus also sets of formulas) of $\mathcal{K}_E$ and of $\mathcal{K}_I$ onto formulas of AELB. We then test for specific sets of formulas from $\mathcal{K}_E$ and from $\mathcal{K}_I$ their images for equivalence in AELB.

Przymusinski showed that the stable semantics of Extended Logic Programs can be embedded into AELB by replacing $notC$ by $\mathcal{B}\neg C$, meaning "C is believed to be false". Alferes and Pereira continued this work on Extended Logic Programming and showed that also $Stab_3(P)$, i.e., the stationary or three-valued stable semantics, and $WF(P)$, i.e., the wellfounded semantics of such programs P can be embedded into AELB by the same replacement. For the embedding one also needs to augment the original objective languages $\mathcal{K}$, $\mathcal{K}_B$, $\mathcal{K}_E$ and $\mathcal{K}_I$ to the languages $\mathcal{K}'$, $\mathcal{K}'_B$, $\mathcal{K}'_E$ and $\mathcal{K}'_I$ by adding new atomic symbols $\overline{A}$ for every atomic symbol A of $\mathcal{K}$ (respectively $\mathcal{K}_B$, $\mathcal{K}_E$ and $\mathcal{K}_I$). To extend the definition of the consequences of a $\mathcal{K}'_B$ theory we further assume for every atomic symbol A the following *explicit negation axioms*:

$$A \supset \mathcal{B}\neg\overline{A} \quad \text{and} \quad \overline{A} \supset \mathcal{B}\neg A \tag{4}$$

The intended meaning of $\overline{A}$ is "$\overline{A}$ is the explicit negation of A". The explicit negation axioms correspond to the following Coherence Principle, see Equation 1, of Alferes and Pereira [17]:

A implies *not* $\overline{A}$ and
$\overline{A}$ implies *not* A

Definition 6 *The set of* consequences $Cn^*(T)$ *of a* $\mathcal{K}'_B$ *theory* T *is defined as the set of logical consequences of the theory* T *augmented with the three axioms (2), (3) and (4):*

$$Cn^*(T) = Cn(T \cup \{(2), (3), (4)\}).$$

The derivability *relation* $\vdash^*$ *is extended to* $\mathcal{K}'_B$ *by* $T \vdash^* F$ *iff* $F \in Cn^*(T)$.

We further define

- If T is a theory of $\mathcal{K}'_B$ and E is a static expansion of T, then

$$E \restriction \mathcal{K}_B = \{F \in E \mid F \text{ is a formula of } \mathcal{K}_B\}.$$

- $E(T) \restriction \mathcal{K}_B = \{E \restriction \mathcal{K}_B \mid E \in E(T)\}$.
- $\mathcal{K}_B$ theory T_1 and $\mathcal{K}'_B$ theory T_2 are *equivalent* iff $E(T_1) = E(T_2) \restriction \mathcal{K}_B$.

The embedding K_{IMEX} of Imex Logic Programming as needed for the comparison of imex negation with the explicit and implicit negation of Extended Logic Programming is defined as follows.

Definition 7 *Let P be an Imex Logic Program in one of the languages $\mathcal{K}_I$ or $\mathcal{K}'_I$. Let A ($\overline{A}$) be an atom and let L_i ($0 \leq i \leq n$) be literals. The operator K_{IMEX} is defined inductively by:*

$$\begin{array}{lcl} K_{IMEX}(L) & := & \begin{cases} L & \text{if } L = A \text{ or } L = \overline{A} \\ \mathcal{B}\neg L & \text{if } L = {\sim}A \text{ or } L = {\sim}\overline{A} \end{cases} \\ K_{IMEX}(L_0 \leftarrow L_1, \ldots, L_n) & := & K_{IMEX}(L_n) \wedge \ldots \wedge K_{IMEX}(L_1) \supset K_{IMEX}(L_0) \\ K_{IMEX}(P) & := & \{K_{IMEX}(r) \mid r \in P\} \end{array}$$

The embedding K_{ELP} of Extended Logic Programming as needed for the comparison of imex negation with the explicit and implicit negation of Extended Logic Programming is defined as follows.

Definition 8 *Let P be a Logic Program in one of the languages $\mathcal{K}_E$ or $\mathcal{K}'_E$. Let A ($\overline{A}$) be an atom and let L_i ($0 \leq i \leq n$) be literals. The operator K_{ELP} is defined inductively by:*

$$\begin{array}{lcl} K_{ELP}(L) & := & \begin{cases} A & \text{if } L = A \\ \mathcal{B}\neg A & \text{if } L = not\, A \\ \overline{A} & \text{if } L = -A \text{ or } L = \overline{A} \\ \mathcal{B}\neg\overline{A} & \text{if } L = not - A \text{ or } L = not\overline{A} \end{cases} \\ K_{ELP}(L_0 \leftarrow L_1, \ldots, L_n) & := & K_{ELP}(L_n) \wedge \ldots \wedge K_{ELP}(L_1) \supset K_{ELP}(L_0) \\ K_{ELP}(P) & := & \{K_{ELP}(r) \mid r \in P\} \end{array}$$

In the translation K_{IMEX} the imex negated statement ${\sim}A$ is always translated to $\mathcal{B}\neg A$. In contrast, the transformation K_{ELP} defined above translates an implicitly negated statement $not A$ by $\mathcal{B}\neg A$. The interaction of explicit negation with the implicit negation is dealt with semantically by the explicit negation axioms that were added to AELB. To circumvent problems arising from this interaction, the well-known technique of translating $-A$ by a new atomic symbol is incorporated into K_{ELP}. The appropriateness of K_{ELP} is given by Przymusinski, Alferes and Pereira as they proved the following theorem.

Theorem 9 *[25, 2] Let P be an Extended Logic Program in the language $\mathcal{K}_E$, then*

- *$Cons_2(K_{ELP}(P))$ corresponds to $Stab_2(P)$ [25].*
- *$Cons_3(K_{ELP}(P))$ corresponds to $Stab_3(P)$ [2].*
- *$Stat(K_{ELP}(P))$ corresponds to $WF(P)$ [2].*

We continue by defining a (K_{ELP}, K_{IMEX})-equivalence relation in terms of the equivalence notion for AELB. Afterwards, we define the translations from the Extended to the Imex Logic Programs and back.

Definition 10 *Let P_1 (respectively P_2) be a set of formulas from a language $\mathcal{L}_1$ (cq. $\mathcal{L}_2$). Furthermore, let m_1 (respectively m_2) be a mapping from $\mathcal{L}_1$ (cq. $\mathcal{L}_2$) in a logic $\mathcal{L}_3$, then P_1 and P_2 are (m_1, m_2)-*equivalent *iff $m_1(P_1)$ and $m_2(P_2)$ are equivalent in $\mathcal{L}_3$. Let $\mathcal{C}_1$ (respectively $\mathcal{C}_2$) be a class of sets of formulas of $\mathcal{L}_1$ (respectively $\mathcal{L}_2$), then $\mathcal{C}_1$ and $\mathcal{C}_2$ are (m_1, m_2)-equivalent iff for every set of formulas P_1 in $\mathcal{C}_1$ there is a set of formulas P_2 in $\mathcal{C}_2$ such that P_1 and P_2 are (m_1, m_2)-equivalent and vice versa.*

Definition 11 *Let P be an Extended Logic Program in the language $\mathcal{K}_E$ and A an atom. We define the operator ψ that translates P into an Imex Logic Program in the language $\mathcal{K}'_I$ inductively by*

$$\begin{array}{lcl} \psi(L) & := & \left\{ \begin{array}{ll} A & \text{if } L = A \\ \sim A & \text{if } L = notA \\ \overline{A} & \text{if } L = -A \\ \sim \overline{A} & \text{if } L = not{-}A \end{array} \right. \\ \psi(L_0 \leftarrow L_1, \ldots, L_n) & := & \psi(L_0) \leftarrow \psi(L_1), \ldots, \psi(L_n) \\ \psi(P) & := & \{\psi(r) \mid r \in P\} \end{array}$$

Although the atomic symbols A and $\overline{A}$ are in general not negatively related in $\psi(P)$, this is not a problem in this article since the Coherence Principle is present in AELB in the form of the explicit negation axioms.

Definition 12 *Let P be an Imex Logic Program in the language $\mathcal{K}_I$. We define the operator ϕ that translates P into an Extended Logic Program in the language $\mathcal{K}_E$ inductively by*

$$\begin{array}{lcl} \phi(L) & := & \left\{ \begin{array}{ll} A & \text{if } L = A \\ notA & \text{if } L = \sim A \end{array} \right. \\ \phi(L_0 \leftarrow L_1, \ldots, L_n) & := & \left\{ \begin{array}{ll} A \leftarrow \phi(L_1), \ldots, \phi(L_n) & \text{if } L_0 = A, \text{ for some atom } A \\ -A \leftarrow \phi(L_1), \ldots, \phi(L_n) & \text{if } L_0 = \sim A, \text{ for some atom } A \end{array} \right. \\ \phi(P) & := & \{\phi(r) \mid r \in P\} \end{array}$$

The rules with a negated atom $\sim A$ as head are those with which the falsity of an atom can explicitly be specified. In Extended Logic Programming that means that $\sim A$ must be translated into $-A$ instead of into $notA$. In the body of Imex Logic Programming rules such a differentiation is impossible but also superfluous. Further note that in $\phi(P)$ the "$-$" symbol does not occur in the bodies of the rules. This essential in the proof of Theorem 14.
In [7] the rules $\{\sim A \leftarrow \overline{A}, \sim\overline{A} \leftarrow A \mid A \in \mathcal{H}_P\}$ that correspond to the Coherence Principle were added to the translation ψ to ensure that the atomic symbols A and $\overline{A}$ are negatively related. The corresponding translation is $\psi'(P) = \{\psi(r) \mid r \in P\} \cup \{\sim A \leftarrow \overline{A}, \sim\overline{A} \leftarrow A \mid A \in \mathcal{H}_P\}$. For analogous reasons, the translation ϕ was extended by the rules $\{-A \leftarrow notA \mid A \in \mathcal{H}_P\}$. So the corresponding transformation ϕ' is defined by $\phi'(P) = \phi(P) \cup \{-A \leftarrow notA \mid A \in \mathcal{H}_P\}$. These rules ensure that the two negations "$-$" and "*not*" of Extended Logic Programming are strongly enough intertwined to compare it with the imex negation. The "$-$" symbol does not occur in the bodies of the rules of $\phi(P)$. Furthermore, we are only interested in the AELB-literals of the form $\mathcal{B}L$. Therefore, these extensions are not needed in this paper, see the proof of Theorem 14. Using the translations ψ' and ϕ', in [7] it has been proved that for every Extended Logic Program P there is an Imex Logic Program P' such that the model-theoretical interpretations of P correspond one to one to the interpretations of P' and vice versa. More specifically, Extended and Imex Logic Programming are equivalent with respect to the stable, stationary, supported and wellfounded Semantics. Furthermore, in [9] it has been proved that Extended and Imex Logic Programming are equivalent with respect to appropriate extensions of Clark's Completion.

Theorem 13 *Let P be an Extended Logic Program in the language $\mathcal{K}_E$, then*

$$K_{ELP}(P) = K_{IMEX}(\psi(P)).$$

Proof by the induction of Definition 11. Let A be an objective atom. For simplicity, we abbreviate

K_{IMEX} to K.

$$
\begin{array}{lcl}
K(\psi(L)) & = & \begin{cases} K(A) = A = K_{ELP}(L) & \text{if } L = A \\ K(\sim A) = \mathcal{B}\neg A = K_{ELP}(L) & \text{if } L = notA \\ K(\overline{A}) = \overline{A} = K_{ELP}(L) & \text{if } L = -A \\ K(\sim\overline{A}) = \mathcal{B}\neg\overline{A} = K_{ELP}(L) & \text{if } L = not{-}A \end{cases} \\
K(\psi(L_0 \leftarrow L_1, \ldots, L_n)) & = & K(\psi(L_n)) \wedge \ldots \wedge K(\psi(L_1)) \supset K(\psi(L_0)) \; = \\
 & & K_{ELP}(L_n) \wedge \ldots \wedge K_{ELP}(L_1) \supset K_{ELP}(L_0) \; = \\
 & & K_{ELP}(L_0 \leftarrow L_1, \ldots, L_n) \\
K(\psi(P)) & = & \{K(\psi(r)) \mid r \in P\} \; = \{K_{ELP}(r) \mid r \in P\} \; = K_{ELP}(P)
\end{array}
$$

□

Theorem 14 *Let P be an Imex Logic Program in the language $\mathcal{K}_I$, then the theories $K_{IMEX}(P)$ and $K_{ELP}(\phi(P))$ are equivalent.*

Proof We define $nlp(P) = \{r \in P \mid hd(r) \text{ is an atom}\}$ and $neg(P) = \{r \in P \mid r \notin nlp(P)\}$. We first show by the induction of Definition 12 that $K_{IMEX}(nlp(P)) = K_{ELP}(\phi(nlp(P)))$:

$$
\begin{array}{lcl}
K_{ELP}(\phi(L)) & = & \begin{cases} K_{ELP}(A) = A = K_{IMEX}(L) & \text{if } L = A \\ K_{ELP}(notA) = \mathcal{B}\neg A = K_{IMEX}(L) & \text{if } L = \sim A \end{cases} \\
K_{ELP}(\phi(A \leftarrow L_1, \ldots, L_n)) & = & K_{ELP}(A \leftarrow \phi(L_1), \ldots, \phi(L_n)) \\
 & = & K_{IMEX}(A \leftarrow L_1, \ldots, L_n) \\
K_{ELP}(\phi(nlp(P))) & = & \{K_{ELP}(\phi(r)) \mid r \in nlp(P)\} \\
 & = & \{K_{IMEX}(r) \mid r \in nlp(P)\} \\
 & = & K_{IMEX}(nlp(P))
\end{array}
$$

We still have to check the rules $\sim A \leftarrow L_1, \ldots, L_n$ in $neg(P)$. For these rules we have

$$\phi(\sim A \leftarrow L_1, \ldots, L_n) = -A \leftarrow \phi(L_1), \ldots, \phi(L_n)$$

and

$$K_{ELP}(-A \leftarrow \phi(L_1), \ldots, \phi(L_n)) = K_{IMEX}(L_n) \wedge \ldots \wedge K_{IMEX}(L_1) \supset \overline{A}.$$

Whereas

$$K_{IMEX}(\sim A \leftarrow L_1, \ldots, L_n) = K_{IMEX}(L_n) \wedge \ldots \wedge K_{IMEX}(L_1) \supset \mathcal{B}\neg A.$$

In other words, $K_{IMEX}(P)$ differs slightly from $K_{ELP}(\phi(P))$. Further note that $K_{IMEX}(P)$ is a theory in $\mathcal{K}_B$ while $K_{ELP}(\phi(P))$ is a theory in $\mathcal{K}'_B$. Therefore, we can only compare the expansions of both theories if we restrict each expansion E of $K_{ELP}(\phi(P))$ to the formulas of $\mathcal{K}_B$. That is, every $\overline{A}$ is ignored. We have proved the theorem if for every formula F of $\mathcal{K}_B$ such that $K_{IMEX}(P) \vdash^* F$ if and only if $K_{ELP}(\phi(P)) \vdash^* F$. Since the Coherence Principle is incorporated as the explicit negation axiom in AELB, we already have that $\overline{A} \supset \mathcal{B}\neg A$. Therefore, we have $K_{IMEX}(P) \vdash^* F$ implies $K_{ELP}(\phi(P)) \vdash^* F$. The other direction is trivially true, since in $\phi(P)$ the "$-$" symbol only occurs on the left-hand side of the rules. That is, any effect statements of the form $-A$ can have on other statements is through the Coherence Principle. Translated into AELB, we see that $-A$ can only influence other statements through the explicit negation axiom. □

Corollary 15 *The classes ELP and IMEX are (K_{ELP}, K_{IMEX})-equivalent.*

This corollary follows directly from the Theorems 13 and 14.

The following corollary holds for three reasons. Firstly, we have the formal connection between the semantics of Extended Logic Programming and AELB that hold under the K_{ELP} mapping as reflected by Theorem 9. Secondly, from [7] we know that ELP and IMEX are semantically equivalent with respect to the stable, stationary and wellfounded semantics. These two steps realise a one-to-one correspondence between the various semantics of an Imex Logic Program P and the specific types of expansions of the AELB-theory $K_{ELP}(\phi(P))$. Thirdly, from Theorems 13 and 14 it follows that this correspondence is induced by K_{IMEX}.

Corollary 16 *Let P be an Imex Logic Program in the language $\mathcal{K}_I$, then*

- $Cons_2(K_{IMEX}(P))$ *corresponds to* $Stab_2(P)$.
- $Cons_3(K_{IMEX}(P))$ *corresponds to* $Stab_3(P)$.
- $Stat(K_{IMEX}(P))$ *corresponds to* $WF(P)$.

6 Discussion

In this paper we studied the interpretations of negations in knowledge representation languages that meet two criteria. The first is that negations can be derived in a direct manner; in a logic programming language this corresponds to the possibility of having negations in the heads of rules. The second criterion is that a nonmonotone derivation mechanism is used to draw negative conclusions; in a logic programming language this corresponds to deriving negative information using some form of CLosed World Assumption. Languages that meet only one of these criteria, like Definite Logic Programming with classical negation and Normal Logic Programming are well studied and it is known how to interpret their negations. For the first criteria the negative conclusions are drawn monotonically, based on the fact that negative statements are allowed in the heads of the rules. For the second several nonmonotonic semantics exist.

However, by combining both direct and indirect inferences of negative conclusions in one complex system, the semantics can no longer be monotone, because of the nonmonotone derivation mechanism of the indirect derivations. On the other hand, it can no longer be the same nonmonotone derivation mechanism that corresponds to the indirect derivations, since there are additional mechanisms to derive falsity. Actually, in derivations in the complex system the direct and the indirect derivations are integrated.

We showed that both Extended and Imex Logic Programming and their semantics provide a logical foundation for the way negations are commonly used in knowledge representation languages. Extended Logic Programming distinguishes in its syntax two different negation symbols, one of which corresponds to the direct derivation of negative information, the other to the indirect derivation. Imex Logic Programming uses one negation symbol and provides all negative conclusion with a uniform semantics.

The intended meanings of the explicit and implicit negations of Extended Logic Programming, namely "proving falsity" and "believed to be not proven", can be formalised by an interpretation mapping into AELB, see [2]. In this paper it was shown that the intended meaning of imex negation, namely "believed to be false" can also be formalised by an interpretation mapping into AELB. For both interpretation mappings the stable, stationary and wellfounded semantics of the programs correspond to specific types of expansions of the AELB-theories. Furthermore, we proved that for every Extended Logic Program P there is an Imex Logic Program P' (and vice versa) such that the embeddings of P and P' into AELB are equivalent AELB-theories.

7 Acknowledgement

The author wishes to thank J.J. Alferes, L. M. Pereira and J. Treur for valuable discussions on the topics of this paper.

References

[1] J.J. Alferes and L.M. Pereira. On logic program semantics with two kinds of negation. In K.Apt, editor, *Logic Programming, Proceedings of the Joint Int. Conf. and Symp. on Logic Programming*, pages 574–588. MIT Press, 1992.

[2] J.J. Alferes and L.M. Pereira. Belief, provability, and logic programs, 1994. LPNMR-Holdings, ftp site: ftp.ms.uky.edu, directory /pub/lpnmr.

[3] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. Fifth International Conference Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[4] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In D.H.D. Warren and P. Szeredi, editors, *7th International Conference on Logic Programming*, pages 579–597. MIT Press, 1990.

[5] H. Herre. Compactness properties of nonmonotonic inference operations. In C. MacNish, D. Pearce, and L.M. Pereira, editors, *Logics in Artificial Intelligence, Proceedings of the fourth European Workshop on Logics in AI, JELIA'94*, volume 838 of *Lecture Notes in Artificial Intelligence*, pages 19–33. Springer-Verlag, 1994.

[6] C.M. Jonker. Implicit and explicit negation in logic programming. Technical report, Department of Philosophy, University of Utrecht, 1993.

[7] C.M. Jonker. *Constraints and Negations in Logic Programming.* PhD thesis, Dept. of Philosophy, Utrecht University, 1994. Quæstiones Infinitæ vol. 10, Dissertation.

[8] C.M. Jonker. Negations in logic programming. In H.C.M. de Swart and L.J.M. Bergmans, editors, *Perspectives on Negation*, pages 91–104. Tilburg University Press, 1995.

[9] C.M. Jonker. Rule-based calculi for logic programs with explicit negation. In B. van der Linder, editor, *Proceedings of the second Dutch/German Workshop on Nonmonotonic Reasoning*, 1995.

[10] R.A. Kowalski and F. Sadri. Logic programs with exceptions. In D.H.D. Warren and P. Szeredi, editors, *Logic Programming, Proceedings 7th International Conference*, pages 598–613. MIT Press, 1990.

[11] I.A.van Langevelde, A.W. Philipsen, and J. Treur. Formal specification of compositional architectures. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence, ECAI'92*, pages 272–276. Chichester: John Wiley and Sons, 1992. Extended version (1991): Technical Report IR-282, AI Group, Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam.

[12] J. McCarthy. Circumscription- a form of non-monotonic reasoning. *Artificial Intelligence*, 13 (1,2):27–39, 1980.

[13] J. Minker and C. Ruiz. Semantics for disjunctive logic programs with explicit and default negation. *Fundamenta Informaticae*, 20:145–192, 1994.

[14] Neuron Data. NEXPERT OBJECT *2.0 Functional Description Manual.* Palo Alto, 1991.

[15] D. Pearce and G. Wagner. Logic programming with strong negation. In P. Schroeder-Heister, editor, *Extensions of Logic Programming*, volume 475 of *lecture notes in artificial intelligence*, pages 311–326. Springer-Verlag, 1991.

[16] L.M. Pereira, 1992. Personal Communication, Berlin.

[17] L.M. Pereira and J.J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *Proceedings 10th European Conference on Artificial Intelligence, ECAI'92*, pages 102–106, 1992.

[18] L.M. Pereira, J.J. Alferes, and J.N. Aparicio. Contradiction removal within well-founded semantics. In A. Nerode, W. Marek, and V.S. Subrahmanian, editors, *First International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 105–119. MIT Press, 1991.

[19] L.M. Pereira, J.J. Alferes, and J.N. Aparicio. Default theory for well founded semantics with explicit negation. In D. Pearce and G. Wagner, editors, *Logics in AI*, volume 633 of *Lecture Notes in Artificial Intelligence*, pages 339–356. Springer-Verlag, 1992.

[20] T.C. Przymusinski. Well-founded semantics coincides with three-valued stable semantics. *Fundamenta Informatica*, XIII(4):445–463, 1990.

[21] T.C. Przymusinski. Autoepistemic logics of closed beliefs and logic programming. In A. Nerode, W. Marek, and V.S. Subrahmanian, editors, *Proceedings of the First International Workshop on Logic Programming and Non-monotonic Reasoning, DOOD'91*, pages 3–20. MIT Press, 1991.

[22] T.C. Przymusinski. Semantics of disjunctive logic programs and deductive databases. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *Proceedings of the Second International Conference on Deductive and Object-Oriented Databases, DOOD'91*, pages 85–107. Springer Verlag, 1991.

[23] T.C. Przymusinski. Stabel semantics for disjunctive programs. *New Generation Computing*, 9:401–424, 1991.

[24] T.C. Przymusinski. A knowledge representation framework based on autoepistemic logic of minimal beliefs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, volume 2, pages 952–958. AAAI Press / The MIT Press, 1994.

[25] T.C. Przymusinski. Static semantics for normal and disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 1994. In print.

[26] Trinzic Corporation. AIONDS *Language Reference, Version 6.4*. Trinzic Corporation, 101 University Ave., Palo Alto, CA 94301, USA, 1994.

[27] G. Wagner. A database needs two kinds of negation. In B. Thalheim, J. Demetrovics, and H.D. Gerhardt, editors, *MFDBS 91*, volume 495 of *Lecture Notes in Computer Science*, pages 357–371. Springer-Verlag, 1991.

[28] G. Wagner. Vivid reasoning with negative information, 1991.

[29] G. Wagner. Neutralization and preemption in extended logic programs. Technical Report Bericht Nr. 20/93, Gruppe für Logic, Wissenstheorie und Information, Freien Universität Berlin, 1993.

A Interpreting $\mathcal{K}_E$ and $\mathcal{K}_I$

A.1 Syntax

We add the propositional constants c, t, u and f to $\mathcal{K}_E$ and $\mathcal{K}_I$ and the propositional connectives $\wedge$ and $\vee$, $\leftarrow$.
We assume the reader to be familiar with the notions "term" and "ground term", "ground atom" and "ground instance of an atom". The *basic formulas* of $\mathcal{L}$ are the atoms plus the propositional constants c, t, u and f. The set $FORM_E$ ($FORM_I$) of *formulas*, denoted by F, G, F_1, G_1, F_2, G_2, ..., of $\mathcal{K}_E$ ($\mathcal{K}_I$) is generated as follows:

1. If F is a basic formula of $\mathcal{K}_E$, then F, $notF$, $-F$ and $not{-}F$ are in $FORM_E$. Similarly, if F is a basic formula of $\mathcal{K}_I$, then F and $\sim F$ are in $FORM_I$.

2. If F and G are elements of $FORM_E$ ($FORM_I$), then $F \wedge G$, $F \vee G$ and $F \leftarrow G$ are in $FORM_E$ ($FORM_I$).

For informal interpretations of the negation symbols of $\mathcal{K}_E$ and $\mathcal{K}_I$, we refer to Section 2. We assume, without loss of generality, that a Logic Program is a possibly infinite set of ground rules. In [3] Gelfond and Lifschitz made the same simplifying assumption. The set $Lit(\mathcal{L})$ is the set of all literals of a language $\mathcal{L}$. Similarly, $Lit(P)$ is the set of all literals of the language underlying the program P.

A.2 Semantics

In this section we define the notions interpretation, valuation and model. We define two orderings on the set of truth-values $\mathcal{FOUR}$ and extend these two orderings to interpretations. The truth-values used in this paper are **f**, for *false*, **u**, for *undefined*, **t**, for *true*, and **c**, for *contradictory*. The set of propositional constants $\{\mathsf{f}, \mathsf{u}, \mathsf{t}, \mathsf{c}\}$ is closely related to $\mathcal{FOUR}$: f corresponds to **f**, u to **u**, t to **t** and c to **c**. We use two different transitive orderings of the truth values as a lattice: the lattice 4_t defined by $\mathbf{f} <_t \mathbf{u} <_t \mathbf{t}$ and $\mathbf{f} <_t \mathbf{c} <_t \mathbf{t}$ (truth-ordering) and the semi-lattice 4_k defined by $\mathbf{u} <_k \mathbf{f} <_k \mathbf{c}$ and $\mathbf{u} <_k \mathbf{t} <_k \mathbf{c}$ (knowledge-ordering). We assume the reader to be familiar with the notions "least upper bound" and "greatest lower bound" with respect to an ordering. These notion are abbreviated by *lub* and *glb*. For $<_t$, we use lub_t and glb_t. The reason we use four truth-values instead of three is that inconsistent programs can occur:

Example 17 Consider the imex program P:

$$\begin{aligned} A &\leftarrow \\ \sim A &\leftarrow \end{aligned}$$

Informally, the rules of P express that both A and $\sim A$ are true, since both rules have an empty body. The truth of $\sim A$ implies the falsity of A, so that A is both true and false. In the same way the truth of A implies the falsity of $\sim A$, so that $\sim A$ is also both true and false. Therefore, A is contradictory and the value **c** will be assigned to it.

In the following we find it convenient to work with the set of literals that are true in an interpretation. We show that there exists a one-to-one correspondence between this set of literals and the interpretation itself. Since we assume that P is a possibly infinite set of ground rules it is sufficient to define interpretations of $\mathcal{K}_E$ and of $\mathcal{K}_I$ as functions from the ground formulas of $\mathcal{K}_E$ (respectively $\mathcal{K}_I$) to $\mathcal{FOUR}$. We assume that $\sim\sim F$ denotes F and $--F$ denotes F, but $not{-}F$ does in general not denote F.

Definition 18 (Interpretation of $\mathcal{K}_E$) An interpretation I of language $\mathcal{K}_E$ is a function from the subset $FORM_E$ of ground formulas of $\mathcal{K}_E$ to the set $\mathcal{FOUR}$ such that for every ground formula F of $\mathcal{K}_E$:

1. If F is a literal L of $\mathcal{K}_E$, then
   - $I(notL) = \mathbf{t}$ iff $I(L) = \mathbf{f}$.
   - $I(notL) = \mathbf{u}$ iff $I(L) = \mathbf{u}$.
   - $I(notL) = \mathbf{c}$ iff $I(L) = \mathbf{c}$.
   - For all $y_1 \in \{\mathsf{f}, \mathsf{u}, \mathsf{t}, \mathsf{c}\} \subseteq FORM_E$ and $y_2 \in \mathcal{FOUR}$: $I(y_1) = I(y_2)$ iff y_1 corresponds to y_2.
2. If the formula F has the form $F_1 \wedge F_2$, then $I(F) = glb_t(I(F_1), I(F_2))$

3. If the formula F has the form $F_1 \vee F_2$, then $I(F) = lub_t(I(F_1), I(F_2))$

4. If the formula F has the form $F_1 \leftarrow F_2$, then

$$I(F) = \begin{cases} \mathbf{t} & \text{iff } I(F_2) \leq_t I(F_1) \\ \mathbf{f} & \text{iff } I(F_2) \not\leq_t I(F_1) \text{ and } I(F_1) \neq \mathbf{c} \text{ and } I(F_2) \neq \mathbf{c} \\ \mathbf{c} & \text{otherwise} \end{cases}$$

Definition 19 (Interpretation of $\mathcal{K}_I$) An interpretation I of language $\mathcal{K}_I$ is a function from the subset $FORM_I$ of ground formulas of $\mathcal{K}_I$ to the set $\mathcal{FOUR}$ such that for every ground formula F of $\mathcal{K}_I$:

1. If F is a literal L of $\mathcal{K}_I$, then
   - $I(\sim L) = \mathbf{t}$ iff $I(L) = \mathbf{f}$.
   - $I(\sim L) = \mathbf{u}$ iff $I(L) = \mathbf{u}$.
   - $I(\sim L) = \mathbf{c}$ iff $I(L) = \mathbf{c}$.
   - For all $y_1 \in \{\mathsf{f}, \mathsf{u}, \mathsf{t}, \mathsf{c}\} \subseteq FORM_E$ and $y_2 \in \mathcal{FOUR}$: $I(y_1) = I(y_2)$ iff y_1 corresponds to y_2.
2. If the formula F has the form $F_1 \wedge F_2$, then $I(F) = glb_t(I(F_1), I(F_2))$
3. If the formula F has the form $F_1 \vee F_2$, then $I(F) = lub_t(I(F_1), I(F_2))$
4. If the formula F has the form $F_1 \leftarrow F_2$, then

$$I(F) = \begin{cases} \mathbf{t} & \text{iff } I(F_2) \leq_t I(F_1) \\ \mathbf{f} & \text{iff } I(F_2) \not\leq_t I(F_1) \text{ and } I(F_1) \neq \mathbf{c} \text{ and } I(F_2) \neq \mathbf{c} \\ \mathbf{c} & \text{otherwise} \end{cases}$$

We say interpretation I of language $\mathcal{L}$ is consistent iff there is no literal L of $\mathcal{L}$ such that $I(L) = \mathbf{c}$. I is two-valued if I is consistent and for every literal L of $\mathcal{L}$ we have $I(L) \neq \mathbf{u}$.
In the definition of interpretation of $\mathcal{K}_E$ the negation symbol "$-$" is treated differently (no restriction is placed on the interpretation of $-A$) from the negation symbol "*not*". In fact, A and $-A$ are still totally unrelated. Therefore, we extend the definition of an interpretation of $\mathcal{K}_E$ by:

Definition 20 (Coherence) [17] Let I be an interpretation of $\mathcal{K}_E$ then I is a *coherent* interpretation of $\mathcal{K}_E$ if for all literals C of $\mathcal{L}$ that are of the form A or $-A$: If $I(C) \in \{\mathbf{t}, \mathbf{c}\}$ then $I(-C) = \mathbf{f}$ or $I(-C) = \mathbf{c}$.

Lemma 21 *If I is a coherent interpretation of $\mathcal{K}_E$, then $I(not{-}C \leftarrow C) = \mathbf{t}$ for all literals C of the form A or $-A$.*

Proof Let C be a literal of the form A or $-A$ and consider the four cases:
• $I(C) = \mathbf{c}$. Since I is coherent, we have $I(C) = \mathbf{c}$ implies that $I(-C) \in \{\mathbf{f}, \mathbf{c}\}$. By definition of interpretation we have $I(-C) \in \{\mathbf{f}, \mathbf{c}\}$ implies that $I(not{-}C) \in \{\mathbf{t}, \mathbf{c}\}$ and thus $I(not{-}C \leftarrow C) = \mathbf{t}$.
• $I(C) = \mathbf{t}$ Since I is coherent, $I(C) = \mathbf{t}$ implies that $I(-C) \in \{\mathbf{f}, \mathbf{c}\}$. By definition of interpretation we have $I(-C) \in \{\mathbf{f}, \mathbf{c}\}$ implies that $I(not{-}C) \in \{\mathbf{t}, \mathbf{c}\}$. Unless $I(not{-}C) = \mathbf{c}$, $I(not{-}C \leftarrow C) = \mathbf{t}$. Suppose $I(not{-}C) = \mathbf{c}$, then, because I is an interpretation, $I(-C) = \mathbf{c}$. Since I is coherent, $I(C) \in \{\mathbf{f}, \mathbf{c}\}$, which contradicts the assumption that $I(C) = \mathbf{t}$.
• $I(C) = \mathbf{u}) \in I$. By definition of interpretation we have that $I(C) \notin \{\mathbf{f}, \mathbf{t}, \mathbf{c}\}$. Suppose $I(not{-}C \leftarrow C) \neq \mathbf{t}$. By definition of interpretation this means that $I(not{-}C) \in \{\mathbf{f}, \mathbf{c}\}$ and thus that $I(-C) \in \{\mathbf{t}, \mathbf{c}\}$. Since I is coherent, $I(-C) \in \{\mathbf{t}, \mathbf{c}\}$ implies $I(--C) \in \{\mathbf{f}, \mathbf{c}\}$ and thus $I(C) \in \{\mathbf{f}, \mathbf{c}\}$. By definition of interpretation $I(C) \in \{\mathbf{f}, \mathbf{c}\}$ contradicts $I(C) = \mathbf{u}$.
• $I(C) = \mathbf{f}$. For all y such that $I(not{-}C) = y$, we have $\mathbf{f} \leq_t y$ and thus $I(not{-}C \leftarrow C) = \mathbf{t}$. □

Definition 22 (Model) Let I be a consistent interpretation of ground language $\mathcal{K}_E$ or $\mathcal{K}_I$, let F be a ground formula of the same language and let S be a set of ground formulas of the language. Then

a. I is a model for F if $I(F) = \mathbf{t}$.

b. I is a model for S if I is a model for each formula of S.

I is a *coherent model* for S of $\mathcal{K}_E$ iff I is a model for S and I is a coherent and consistent interpretation of $\mathcal{K}_E$.

Example 23 The influence of coherence can be seen by considering the set S consisting of the formulas:

$$\begin{array}{rcl} A_1 & \leftarrow & A_2 \\ -A_2 & \leftarrow & not{-}A_2 \end{array}$$

There exists a model I of S such that $I(A_2) = \mathbf{t}$ and $I(-A_2) = \mathbf{t}$, which makes I not coherent. In this model $I(A_1) = \mathbf{t}$ as well. In a coherent model A_2 and $-A_2$ cannot both be true. Although the coherence principle relates C and $-C$, the coherence principle does not imply that for every coherent interpretation I': $I'(C) = \mathbf{t}$ iff $I'(-C) = \mathbf{f}$ for every C. There exists, for example, a model I_1 for S such that $I_1(A_2) = \mathbf{f}$ and $I_1(-A_2) = \mathbf{u}$. In this model we also have $I_1(A_1) = \mathbf{f}$. □