Логические исследования
2021. Т. 27. № 1. С. 64–81
УДК 16+510.6

Logical Investigations
2021, Vol. 27, No. 1, pp. 64–81
DOI: 10.21146/2074-1472-2021-27-1-64-81

Reinhard Kahle

Default Negation as Explicit Negation plus Update

Reinhard Kahle
Universität Tübingen,
Theorie und Geschichte der Wissenschaften, Keplerstr.
2, D-72074 Tübingen, Germany.
Centro de Matemática e Aplicações,
FCT, Universidade Nova de Lisboa, P-2829-516 Caparica, Portugal.
E-mail: kahle@mat.uc.pt

Abstract: We argue that under the stable model semantics default negation can be read as explicit negation with update. We show that dynamic logic programming which is based on default negation, even in the heads, can be interpreted in a variant of updates with explicit negation only. As corollaries, we get an easy description of default negation in generalized and normal logic programming where initially negated literals are updated. These results are discussed with respect to the understanding of negation in logic programming.

Keywords: Default Negation, Explicit Negation, Logic Programming Update

For citation: Kahle R. "Default Negation as Explicit Negation plus Update", *Logicheskie Issledovaniya / Logical Investigations*, 2021, Vol. 27, No. 1, pp. 64–81. DOI: 10.21146/2074-1472-2021-27-1-64-81

1. Introduction

Negation is still one of the controversial concepts underlying logic programming. While the *negation-as-failure* interpretation is operationally well-understood, the logical interpretation of negation gives rise to discussions. Starting from the two most prominent approaches – Reiter's *closed world assumption* [Reiter, 1978] and Clark's *completion* [Clark, 1978] – there "are very successful attempts to discover the underlying logic of negation as failure. Their disadvantage is that the logics involved are more complicated and less familiar than classical logic so that they are not likely to help the naive programmer express his problem by means of a logic program, or to check the correctness of a program" [Shepherdson, 1998, p. 364]. In this paper we argue that *default* negation in logic programming can be understand as *explicit* negation in an *update* framework, as long as we consider stable model semantics. This reading may also give a "naive programmer" some help to deal with default negation in logic programming.

The question of treating updates in a logic programming framework is a research project in itself. There are several attempts to deal with updates and different semantics have been proposed, based on stable models or answer sets, cf. e.g. [Alferes et al., 2000; Buccafurri et al., 1999; Eiter et al., 2002; Leite and Pereira, 1998a; Leite and Pereira, 1998b; Leite, 2003; Sakama and Inoue, 1999; Zhang and Foo, 1998]. In all these frameworks updates are represented by a sequence of logic programs. Here, we focus on *dynamic logic programming*, introduced by Alferes, Leite, Pereira, Przymusinska, and Przymusinski [Alferes et al., 2000]. It is based on *generalized logic programs* which allow *default negation* in the head of rules. Its semantics is based on causal rejection, i.e., a rule can be rejected if there is a more recent one that conflicts with it.

We show that *default negation* as used in dynamic logic programming, generalized logic programming, and normal logic programming can be treated as *explicit negation* in an analogous update framework. For the technical result we can build on work of Leite in his dissertation [Leite, 2003]. However, with respect to the understanding of default negation, the result allows a different perspective to it. In particular, it questions the status of default negation as a special form of negation, different from the classical one. In contrast, default negation can be seen as involving an update aspect (which could also be considered as a temporal aspect), which can be expressed in an update framework with explicit negation.

In the following section we introduce the technical preliminaries for logic programs with default and with explicit negation. In the third section we introduce an update framework for explicit negation. With it we can state our main result in section 4, which also provides a short illustrating example. In section 5 we review dynamic logic programming, as it is defined in [Alferes et al., 2000] and [Leite, 2003]. It is used in the sixth section to prove a general translation of dynamic logic programs into the explicit update framework. From it, the main result follows as an immediate corollary. The final section is devoted to a discussion of the given result with respect to the closed world assumption, normal logic programming and the combination of default and explicit negation. We shortly address the question of well-founded semantics and the relation to the transformational semantics for dynamic logic programming and include a reference to abductive frameworks.

2. Preliminaries

In logic programming, for both, default and explicit negation, it is convenient to work syntactically with pure Horn theories. In this case, negated literals are introduced as new atoms (disjoint from all other atoms) which are formally positive. The difference of positive and negative literals is build in on the

semantical level only. Thus, for default negation we will use atoms `not_a`, for explicit negation atoms `neg_a`.

Here, we do not consider programs which combine default and explicit negation. Let us first define the technicalities for the case of default negation.

2.1. Default negation

Given an arbitrary set $\mathcal{K}$ of propositional variables (which do not begin with "`not`"), the propositional language $\mathcal{L}^{\mathcal{K}}_{\texttt{not}}$ is defined as the set $\{\texttt{a} : \texttt{a} \in \mathcal{K}\} \cup \{\texttt{not\_a} : \texttt{a} \in \mathcal{K}\}$. Elements of $\mathcal{L}^{\mathcal{K}}_{\texttt{not}}$ are called *literals*, *positive literals* if they belong to $\mathcal{K}$, *negative literals* otherwise. While *normal logic programs* allow default negation only in the body of a clause, we consider *generalized logic programs* where default negation is also allowed in the heads of rules. Generalized logic programs, introduced in [Alferes et al., 2000], are a simplified version of the programs introduced by Lifschitz and Woo [Lifschitz and Woo, 1992]. Formally, a generalized logic program consists of a (finite or infinite) set of rules of the form L `<-` $L_1\texttt{,}\ldots\texttt{,}L_n$, where $L, L_1, \ldots, L_n$ are literals from $\mathcal{L}^{\mathcal{K}}_{\texttt{not}}$.

We use the usual notational conventions in logic programming. If `a` is a positive literal, *not* `a` is `not_a`, and for a negative literal `not_b`, *not* `not_b` is `b`. If r is the rule L `<-` $L_1\texttt{,}\ldots\texttt{,}L_n$ we write $\mathsf{H}(r)$ for the head L and $\mathsf{B}(r)$ for the body $L_1\texttt{,}\ldots\texttt{,}L_n$.

A *2-valued interpretation* M of $\mathcal{L}^{\mathcal{K}}_{\texttt{not}}$ is a subset of $\mathcal{L}^{\mathcal{K}}_{\texttt{not}}$ such that for all $\texttt{a} \in \mathcal{K}$ precisely one of the literals `a` or `not_a` belong to M. An interpretation M of $\mathcal{L}^{\mathcal{K}}_{\texttt{not}}$ satisfies a rule r if $\mathsf{H}(r)$ belong to M, or some literal in $\mathsf{B}(r)$ does not belong to M. A *model* M of a generalized logic program P is an interpretation which satisfies all its rules. Let M^+ be the set of positive literals in M, and M^- the set of negative literals in M. A model N is *p-smaller* than M, if N^+ is a proper subset of M^+, $N^+ \subset M^+$. A model of P is called *p-least* if it is the p-smallest model of P.[1] Now, the definition of stable models reads as follows, cf. [Alferes et al., 2000, Def. 2.1].

Definition 1. An interpretation M of $\mathcal{L}^{\mathcal{K}}_{\texttt{not}}$ is a *stable model* of a generalized logic program P if M is the p-least model of the Horn theory $P \cup M^-$, i.e.,

$$M = \textit{p-least}(P \cup M^-).$$

For normal programs this definition is equivalent to the original definition of stable models by Gelfond and Lifschitz [Gelfond and Lifschitz, 1988]. In [Alferes et al., 2000] it is shown that this definition coincides with the answer set semantics given by Lifschitz and Woo, [Lifschitz and Woo, 1992], if it is restricted to generalized logic programs.

[1] We say "p-least", since we consider the positive literals, only. Later on, for explicit negation, we will define *leastness* with respect to all literals.

2.2. Explicit negation

Extended logic programs are well-known as the extension of normal logic programs with explicit negation. The *answer-set semantics* given for this class of programs allows the use of explicit negation in both, the body and the head of a clause, [Gelfond and Lifschitz, 1991].

As for the default literals, explicit negated literals are introduced as new atoms, and *consistency* (as well as *coherency* in the presence of default negation) is treated on the semantical level.

Here, we even dispense with default negation, and consider programs with explicit negation only. For convenience we tactically call them *explicit logic programs*. This class of programs, in itself, is without particular interest. But, in connection with update it allows us to give an easy interpretation of default negation.

The formal preliminaries are completely analogous to the case of dynamic logic programming. The only difference is that we replace the "prefix" `not` for default negation by `neg` for explicit negation.

Given a set of propositional variables $\mathcal{K}$, we consider now the language $\mathcal{L}^{\mathcal{K}}_{\mathtt{neg}} = \{\mathtt{a} : \mathtt{a} \in \mathcal{K}\} \cup \{\mathtt{neg\_a} : \mathtt{a} \in \mathcal{K}\}$. As before, we will speak of positive and negative literals, and only if needed we speak about *default negated literals* or *explicit negated literals*. Instead of *not* `a`, which we used for default negation, we write *neg* `a` for the complementary literal with respect to explicit negation.

The notion of model differs from the default case. Since we cannot assume negative literals "by default" we have to ask for support of them as for positive literals. In the default case, we consider least models with respect to the positive literals only. Now, we have to consider least models with respect to both, positive and negative literals. Therefore, we start with consistent interpretations, which do not have to be 2-valued. Thus, an interpretation M is an subset of $\mathcal{L}^{\mathcal{K}}_{\mathtt{neg}}$ such that for no literal $\mathtt{a} \in \mathcal{K}$, `a` and `neg_a` belong to M. With the usual notion of satisfiability we define a *pre-model* M of an explicit logic program which is an interpretation satisfying all rules of the program. It is called "pre-model" since later we are interested in 2-valued models only. We order the pre-models with respect to both, positive and negative literals. A model N is *smaller* than M, if it is a proper subset of M. A pre-model of P is called *least* if it is a smallest pre-model of P. A *2-model* M of an explicit logic program is a least pre-model which is 2-valued with respect to explicit negation, i.e., M contains for all $\mathtt{a} \in \mathcal{K}$ precisely one of the literals `a` or `neg_a`.

Definition 2. An interpretation M of $\mathcal{L}^{\mathcal{K}}_{\mathtt{neg}}$ is the *2-model* of an explicit logic program P if M is 2-valued and the least pre-model of P, i.e., $M = \mathit{least}(P)$.

With the given definition the very most of explicit logic programs will not have a 2-model. As a very easy example, let us consider the generalized logic program P: `not_a <- not_a` and the explicit logic program E: `neg_a <- neg_a`. P has {`not_a`} as single stable model, while E does not have a 2-model. In general, for explicit logic programs, it would be more natural to consider a three valued semantics. However, for our specific aim to interprete default negation in an update framework with explicit negation, it turns out that the existence of 2-models is always guaranteed.

3. Explicit dynamic logic programs

In this section we will give an adaptation of *dynamic logic programming* as introduced in [Alferes et al., 2000] to explicit logic programs.[2]

An *explicit dynamic logic program* $\mathcal{E}$ is a finite sequence of *explicit logic programs* $E_1, \ldots, E_n$, written as

$$E_1 \otimes E_2 \otimes \cdots \otimes E_n.$$

Its semantics is – in analogy to dynamic logic programming – based on causal rejection: A rule can be rejected by a more recent one if the latter rule has as head the negated literal of the former one, and the body of the latter one is true in the considered model.

Formally we introduce the following notion of conflicting rules:

Definition 3. Two rules r and r' are called *conflicting with respect to explicit negation*, denoted by $r \overset{\text{neg}}{\bowtie} r'$ iff $\mathsf{H}(r) = neg\ \mathsf{H}(r')$.

Let $\bigcup \mathcal{E}$ be the union of all rules of all explicit logic programs of an explicit dynamic logic program. Then, we define 2-model of $\mathcal{E}$ as follows:

Definition 4. Let $\mathcal{E} = \{E_i : 0 \leq i \leq n\}$ be an explicit dynamic logic program. An interpretation M is a *2-model of* $\mathcal{E}$, if

1. M is 2-valued and
2. $M = least((\bigcup \mathcal{E}) \setminus Reject(\mathcal{E}, M))$,
   where $Reject(\mathcal{E}, M)$ is the set

$$\{r \in E_i : \exists r' \in E_j, i < j \leq n \,\&\, r \overset{\text{neg}}{\bowtie} r' \,\&\, M \models \mathsf{B}(r')\}.$$

[2] We will review dynamic logic programming only later in Section 5. It is not needed to state the main result of this paper (Proposition 1), but it will be instrumental for the proof of it.

4. Default Negation as Explicit Negation plus Update

We now state the main proposition of our paper. A generalized logic program P can be translated into an explicit dynamic logic program $\mathcal{E}$ such that the stable models of P coincide with the 2-models of $\mathcal{E}$. That means that the sets of positive literals in both sets are the same, and the default negated literals of a model of $\mathcal{P}$ coincide with the explicit negated literals in the corresponding model of $\mathcal{E}$. In fact, $\mathcal{E}$ is an explicit dynamic logic program with one update only.[3]

Definition 5. Let $\mathcal{K}$ be a set of propositional variables.

1. E_0 is defined as the set of all explicit negated literals in the language $\mathcal{K}$:

$$E_0 := \{\texttt{neg\_a} : \texttt{a} \in \mathcal{K}\}.$$

2. Given a generalized logic program P in $\mathcal{L}^{\mathcal{K}}_{\texttt{not}}$, we define the explicit logic program $E = \overline{P}$ in $\mathcal{L}^{\mathcal{K}}_{\texttt{neg}}$ as the program where every occurrence of a default negation in P is replaced by an explicit negation.

3. For a set S of literals of $\mathcal{L}^{\mathcal{K}}_{\texttt{not}}$, we define $\overline{S}$ as the set of literals of $\mathcal{L}^{\mathcal{K}}_{\texttt{neg}}$ where all default negated literals are replaced by their corresponding explicit negated ones.

Of course, E_0 has the trivial 2-model where all negative literals are true. Therefore, when we start an explicit dynamic logic program with the initial program E_0, we guarantee the 2-valuedness of its model.

Proposition 1. Let P be a generalized logic program in the language $\mathcal{K}$. Let $\mathcal{E}$ be the explicit dynamic logic program $E_0 \otimes \overline{P}$. Then,

$$M \text{ is a stable model of } P \text{ if and only if } \overline{M} \text{ is a 2-model of } \mathcal{E}.$$

Analogously we have for normal logic programs the following proposition:

Proposition 2. Let P be a normal logic program in the language $\mathcal{K}$. Let $\mathcal{E}$ be the explicit dynamic logic program $E_0 \otimes \overline{P}$. Then,

$$M \text{ is a stable model of } P \text{ if and only if } \overline{M} \text{ is a 2-model of } \mathcal{E},$$

and $\overline{P}$ does not use negation in the heads.

These propositions are corollaries of Theorem 1 which will be stated and proven below.

[3] To some extend, the study of a single update (instead of sequences of updates) has its own interest. For instance, in the original definition of dynamic logic programming, [Alferes et al., 2000], the authors even start with the definition of one update, and define dynamic logic programs as a generalization of it.

Example

To illustrate our result, let us consider the following normal logic program:

$$P = \{\texttt{a <- not\_b}, \texttt{b <- not\_a}\}.$$

It has the two stable models $M_1 = \{\texttt{a}, \texttt{not\_b}\}$ and $M_2 = \{\texttt{b}, \texttt{not\_a}\}$. The translation of P into an explicit dynamic logic programs yields

$$\mathcal{E} = \{\texttt{neg\_a}, \texttt{neg\_b}\} \otimes \{\texttt{a <- neg\_b}, \texttt{b <- neg\_a}\}.$$

Now, we have to check whether $\overline{M_1} = \{\texttt{a}, \texttt{neg\_b}\}$ and $\overline{M_2} = \{\texttt{b}, \texttt{neg\_a}\}$ are the only 2-models of $\mathcal{E}$ according to Definition 4. Of course, both are 2-valued.

For $\overline{M_1}$, we have that $\mathit{Reject}(\mathcal{E}, \overline{M_1}) = \{\texttt{neg\_a}\}$, since it is rejected by the rule `a <- neg_b` whose body is true in $\overline{M_1}$. Therefore, $\overline{M_1}$ has to be the least pre-model of $\{\texttt{neg\_b}, \texttt{a <- neg\_b}, \texttt{b <- neg\_a}\}$ which is the case.

Analogously, for $\overline{M_2}$, `neg_b` is rejected, and we get that it is the least pre-model of $\{\texttt{neg\_a}, \texttt{a <- neg\_b}, \texttt{b <- neg\_a}\}$.

There are only two other possibilities for 2-models of $\mathcal{E}$, $N_1 = \{\texttt{a}, \texttt{b}\}$ and $N_2 = \{\texttt{neg\_a}, \texttt{neg\_b}\}$. For N_1, $\mathit{Reject}(\mathcal{E}, N_1)$ is empty, so `neg_a` and `neg_b` are facts, and N_1 cannot be a 2-model. For N_2 the situation is different, since $\mathit{Reject}(\mathcal{E}, N_2) = \{\texttt{neg\_a}, \texttt{neg\_b}\}$. But the least pre-model of $\{\texttt{a <- neg\_b}, \texttt{b <- neg\_a}\}$ is the empty set, i.e., this program does not have a 2-model, in particular not N_2. Thus, $\overline{M_1}$ and $\overline{M_2}$ are the only 2-models of $\mathcal{E}$.

5. Dynamic logic programming

For the proof of the Propositions 1 and 2 we will use a more general result, translating dynamic logic programs into explicit dynamic logic programs. Therefore, we review briefly the definition of dynamic logic programs.

A *dynamic logic program* $\mathcal{P}$ consists of a finite sequence of generalized logic programs $P_1, \ldots, P_n$, written as

$$P_1 \oplus P_2 \oplus \cdots \oplus P_n.$$

As for the 2-models of extended dynamic logic programming, the *stable model semantics* of a dynamic logic program is based on causal rejection. The idea of default negation is build in by assuming all negated literals for which there is no rule with the positive literal as head and a true body.

For the formal definition, we need again the notion of conflicting rules, cf. [Leite, 2003, Def. 27, p. 35]:

Definition 6. Two rules r and r' are called *conflicting*, denoted by $r \bowtie r'$, iff $\mathsf{H}(r) = \mathit{not}\ \mathsf{H}(r')$.

Using $\bigcup \mathcal{P}$ for the union of all rules of all generalized logic programs of a dynamic logic program $\mathcal{P}$, we can define stable models of $\mathcal{P}$ as follows, cf. [Leite, 2003, Def. 37, p. 48].

Definition 7. Let $\mathcal{P} = \{P_i : 1 \leq i \leq n\}$ be a dynamic logic program. An interpretation M is a *stable model of* $\mathcal{P}$, if

$$M = p\text{-}least([(\bigcup \mathcal{P}) \setminus Reject(\mathcal{P}, M)] \cup Default(\mathcal{P}, M)),$$

where $Reject(\mathcal{P}, M)$ is the set

$$\{r \in P_i : \exists r' \in P_j, i < j \leq n \,\&\, r \bowtie r' \,\&\, M \models \mathsf{B}(r')\},$$

and $Default(\mathcal{P}, M)$ is the set

$$\{\texttt{not\_a} : \neg\exists r \in \bigcup \mathcal{P}.(\mathsf{H}(r) = \mathtt{a}) \,\&\, M \models \mathsf{B}(r)\}.$$

6. Embedding of dynamic logic programming in explicit dynamic logic programming

We now extend the translation of generalized logic programs in extended dynamic logic programs to dynamic logic programs. With the notation of Definition 5 we set:

Definition 8. For a dynamic logic program $\mathcal{P} = P_1 \oplus \ldots \oplus P_n$, $\overline{\mathcal{P}}$ is defined as $\overline{P_1} \otimes \cdots \otimes \overline{P_n}$.

The general theorem can now be stated as follows:

Theorem 1. *Let $\mathcal{K}$ be a set of propositional variables. Let $\mathcal{P} = P_1 \oplus \ldots \oplus P_n$ be a dynamic logic program in $\mathcal{L}^{\mathcal{K}}_{\mathtt{not}}$. Let $\mathcal{E}$ be the explicit dynamic logic program $E_0 \otimes \overline{P_1} \otimes \cdots \otimes \overline{P_n}$. Then,*

M is a stable model of $\mathcal{P}$ if and only if $\overline{M}$ is a 2-model of $\mathcal{E}$.

The theorem is a consequence of the following proposition, proven by Leite in his dissertation, [Leite, 2003, Prop. 28 and Cor. 29, p. 54].

Proposition 3. Let $\mathcal{P}$ be a dynamic logic program. Let $\mathcal{P}'$ be the dynamic logic program such that $\mathcal{P}' = P^{\mathcal{K}} \oplus P^{not\ \mathcal{K}} \oplus \mathcal{P}$. An interpretation M is a stable model of $\mathcal{P}$ iff

$$M = p\text{-}least((\bigcup \mathcal{P}') \setminus Reject(\mathcal{P}', M)),$$

where $P^{\mathcal{K}}$ is the set of all positive literals in the language $\mathcal{K}$ and $P^{not\ \mathcal{K}}$ the set of all default negated literals in the language $\mathcal{K}$.

Proof. (Theorem 1) We will use Proposition 3. However, the use of $P^{\mathcal{K}}$ is redundant, since $P^{not\ \mathcal{K}}$ is immediately rejecting all literals of $P^{\mathcal{K}}$. Therefore, we can choose for $\mathcal{P}'$ also the dynamic logic program $P^{not\ \mathcal{K}} \oplus \mathcal{P}$. So, $\mathcal{E} = \overline{\mathcal{P}'}$.

The assertion of the theorem seems to be a notational variant where default negation is replaced by explicit negation. However, since dynamic logic programming was formulated by use of p-least models we have to check that the least pre-models used for the semantics in explicit dynamic logic programming are coincide with them, modulo the substitution of default and explicit negation.

For the direction from the left to the right, let M be a stable model of $\mathcal{P}$. With the proposition we get that M is the p-least model of $P_M := (\bigcup \mathcal{P}') \setminus \mathit{Reject}(\mathcal{P}', M)$. Translating this program into an explicit logic program, we get that $\overline{M}$ is a 2-valued pre-model of $E_M := (\bigcup \mathcal{E}) \setminus \mathit{Reject}(\mathcal{P}', \overline{M})$. We have to show that it is the least pre-model. Assume there is another pre-model N of E_M, with $N \subset \overline{M}$. If N^+ is a proper subset of $\overline{M}^+$, then there is a model K of P_M with $N = \overline{K}$. So K^+ is a subset of M^+ which contradicts the assumption that M was the p-least model of P_M. Now, let N^- be a proper subset of $\overline{M}^-$. So there is a literal `neg_b` in $\overline{M}$ which is not in $\overline{N}$. Since E_0 contains the fact `neg_b`, there has to be a rule r in $\mathit{Reject}(\mathcal{P}', \overline{M})$ with $\mathsf{H}(r) = \mathtt{b}$ and the body of r is true in $\overline{M}$. However, with this condition $\overline{M}$ could not be a 2-model of E_M since it has to contain both `neg_b` and `b`. Thus, we have a contradiction, and $\overline{M}$ is the least pre-model of E_M, i.e., $\overline{M}$ is a 2-model of $\mathcal{E}$.

For the direction from the right to the left we start with a 2-model N of $\mathcal{E}$. So N is the least pre-model of $E_N := (\bigcup \mathcal{E}) \setminus \mathit{Reject}(\mathcal{E}, N)$. Of course, there is an interpretation M of $\mathcal{L}^{\mathcal{K}}_{\mathtt{not}}$ such that $N = \overline{M}$. We have to show that M is a stable model of $\mathcal{P}$. Using the proposition above, we have to show that M is a stable model of $P_M := (\bigcup \mathcal{P}') \setminus \mathit{Reject}(\mathcal{P}', M)$. Since $E_N = \overline{P_M}$, and N is a pre-model of E_N, M is a model of P_M. It remains to show that M is p-least. Assume that there is another model K of P_M with $K^+ \subset M^+$. Without loss of generality, we can assume that $K = M \setminus \{\mathtt{b}\}$ for some positive literal `b`. Since K is a model of P_M, there is no rule r in P_M with $\mathsf{H}(r) = \mathtt{b}$ and the body of r is true in K. But that means, there is no rule r' in E_N such that $\mathsf{H}(r') = \mathtt{b}$ and the body of r' is true in $\overline{K} = N \setminus \{\mathtt{b}\}$. Therefore, $N \setminus \{\mathtt{b}\}$ is a pre-model of E_N which contradicts the assumption that N was a 2-model of $\mathcal{E}$. Thus, M is the p-least model of P_M, i.e., M is a stable model of $\mathcal{P}'$ and $\mathcal{P}$. ■

7. Discussion

In this section we discuss our reading of default negation as explicit negation plus update in different respects.

7.1. The closed world assumption

Our explanation of default negation has a certain relation to the closed world assumption. In the closed world assumption "only" those negative literals are assumed for which the positive literal is not derivable from a program. In contrast, we assume all negative literals and update later on those for which the positive literal holds in a model. By this update step we avoid the inconsistencies which result sometimes from the closed world assumption in the case of indefinite information about ground literals, cf. [Shepherdson, 1998, p. 357].

7.2. Default negation in normal logic programming

Proposition 2 is more than a by-product of Theorem 1. It gives the default negation as it is used in normal logic programming a reading in terms of explicit negation with updates. Normal logic programming is the basis of logic programming, uncontroversial and well-understood. However, explicit negation seems to be closer to classical negation as used in standard logic, and therefore more favorable in discussions outside the logic programming community.[4] In fact, it is interesting to study the exact *logical* meanings of negation in logic programming, as it is done in the work of Pearce and others, cf. [Pearce, 1997; Pearce, 1999; Lifschitz et al., 2001]. Here, we have implemented explicitly the idea of defaults as a sceptical view of truth: Every literal, for which one cannot find a rule with a true body, is considered as false. In our reading just the perspective is changed: Every literal is initially considered as false (by use of the initial program E_0) and then we think of the original program as an update program which updates the initial scepticism.

There is a well-known correspondence, cf. [Bidoit and Froidevaux, 1991], between the stable model semantics of normal logic programs and *default logic*, [Reiter, 1980; Poole, 1994]. Now, we can even ask whether the reading of default as explicit plus updates allows for an interpretation of default reasoning as classical reasoning with *updates*. However, this question is outside the scope of this paper.

7.3. The combination of default and explicit negation

A naive extension of our translation does not work when we allow default and explicit negation together.

If one would translate both negations into explicit negation in explicit dynamic logic programming and keep the initial program E_0 in the way that it contains all negative literals, it would trivialize the difference between default and explicit negation. In fact, E_0 would give the original explicit negation the same meaning as default negation.

[4] But see the remarks about explicit negation in the following subsection.

Of course, one could think of restricting E_0 to the negative literals coming from default negation only. However, then, we will have problems to find 2-models for the resulting explicit dynamic logic program, if we have no support for neither `a` nor `neg_a`.

In fact, the understanding of *explicit negation* in logic programming is contrary to the idea of a 2-valued semantics: Only if we explicit information about `a` or `neg_a` we should accept either of it; if not, `a` should be considered as neither true or false. Thus, although explicit negation is in its operational behavior (as it should be guaranteed by the semantics) closer to "classical" negation,[5] it does not support the "classical" principle of bivalence. This principle is better supported by default negation, which guarantees the existence of 2-valued models.

In our translation of default negation in terms of explicit negation, we combine the two "classical" aspects: Since we use explicit negation, its operational behavior is classical; since we start with initial program E_0 which contains all negative literals, we have the 2-valuedness guaranteed. And, the non-monotonic aspect of default negation is resolved in the update step.

7.4. Default and explicit negation in dynamic logic programming

The original formulation of dynamic logic programming in [Alferes et al., 2000] is based on default negation only. But it is argued that the addition of explicit negation to dynamic logic programming is easy, cf. [Ibid., Sec. 5.2], [Leite, 2003, p. 68]. The rejection of rules is still carried out by default negation only. Therefore, *strong negation rules* `not_a <- neg_a` and `not_neg_a <- a` are added which propagate the explicit negation to default negation.

We do not treat this combinations for the reasons given in the preceding subsection.[6] But we like to note, that the use of default negation together with explicit negation gives dynamic logic programming interesting expressive power. In fact, when Leite claims that "logic program updates constitute the *killer application* for generalized logic programs" [Leite, 2003, p. 20], i.e., for the use of default negation in the heads, he uses an example which makes essential use of both, default and explicit negation.[7]

Here, we will not discuss the question of the meaning of default negation in the heads, as used in dynamic logic programming. However, obviously, the

[5] It is a separate discussion "how classical" explicit negation is at the end. What we mean when we say that its operational behavior is closer to classical negation (than the one of default negation) is that we require *only* the consistency for explicit negation and nothing else; in particular it does not involve a non-monotonic aspect.

[6] See [Slota et al., 2014] for a discussion of the combination; this paper also contains additional references to the literature.

[7] The example is also given in [Slota et al., 2014, Example 1].

proposed reading relates default negation in the heads to explicit negation, just with the extra aspect of updates. Maybe, this could be used as an additional justification for generalized logic programs. As general references, aside from [Alferes et al., 2000] and [Lifschitz and Woo, 1992] which were already mentioned above, we can give [Inoue and Sakama, 1998] and [Damásio and Pereira, 1996]. However, the last reference deals mainly with the well-founded semantics instead of the stable model semantics used here.

We should also mention that the treatment of default negation in the presence of explicit negation allows for variations. They relate to the question how the set of rejected literals is defined, cf. e.g., [Leite, 1997; Leite, 2003], and, for a comparison, [Leite, 2004].

As related work in this direction we like to mention the alternative approach to dynamic updates proposed by Eiter et al. [Eiter et al., 2002]. It is based on extended logic programming, i.e., it allows both negations, but only explicit negation in the heads. The authors give a detailed discussion of the relation to dynamic logic programming in the sense of [Alferes et al., 2000], cf. [Eiter et al., 2002, Sec. 7.3].

For the question of combining different forms of negation, the work of Jonker [Jonker, 1994] could be also of interest. She introduces a new form of negation, called *imex* negation which combines aspects of *im*plicit (default) and *ex*plicit negation. It is open whether updates based on this negation would yield different results.

7.5. Stable models versus well-founded semantics

The given reading of default negation in terms of explicit negation plus update is based on the stable model semantics for the default case. It suggests itself to ask how the situation is in the case of *well-founded semantics*, [Gelder, et al., 1991]. This question is open. A well-founded semantics for dynamic logic programs was proposed by Banti, Alferes and Brogi, [Banti et al., 2004], and it could serve as a starting point.

7.6. The transformational semantics

In contrast to [Leite, 2003], in [Alferes et al., 2000] dynamic logic programming is introduced via a *transformational semantics*. In this case (which is equivalent to the declarative semantics given above, cf. [Leite, 2003, Th. 40, p. 66]) a dynamic logic program is translated in a generalized logic program which is formulated in an extended language. It provides for every atom $\mathtt{a}$ a new one $\mathtt{a}^-$ for its explicit negation, and two pairs of them indexed by every separate generalized program and indexed by the stage of the program. Here, we do not give the (longish, but not complicated) definition of the transformational semantics, but just point out that it starts with an *initial state* 0 in which

all positive literals are declared to be false in terms of the new negative atom. It is given by *default rules*: $\mathtt{a}_0^-$ for all positive literals of the language. This initial state can serve as a motivation for the definition of E_0 in our translation above.

7.7. Negation as failure as abduction

There is at least a conceptional relation between our reading and the treatment of logic programs as *abductive frameworks*, cf. [Kakas et al., 1998, Ch. 4; in particular 4.1]. An abductive framework $\langle P, A, I\rangle$ consisting of a logic program P, a set of *abducibles* A, and integrity constraints I. Given a query $\mathtt{q}$, one tries to find one (or more) subset(s) Δ of A such that $P \cup \Delta \models \mathtt{q}$ and $P \cup \Delta$ satisfies I. A logic program with default negation can be translated into an abductive framework where A contains the negated literals; technically, one does not work with the negated literals themselves, but replaces them by new symbols, such that the related system is entirely positive; I contains constraints such that the new literals are correct and complete with respect to negation, cf. [Kakas et al., 1998, p. 255f]. It is a result by Eshghi and Kowalski [Eshghi and Kowalski, 1989] that there is a one to one correspondence between stable models of a logic program and the abductive extensions of its abductive framework. *A fortiori*, our reading can also be related to the abductive framework. Somehow, we just assume all possible abducibles in E_0, and the update step rejects those which can not be used in a particular stable model. But, of course, the remaining set of negative literals could be bigger than the ones needed in a Δ. Thus, abduction allows for a finer analysis of the negative information needed to derive something.

7.8. Logical properties

The interpretation of default negation as explicit negation with update carries over the logical properties of default negation to the use of explicit negation in an update of E_0. Contraposition, for example, does not hold for default negation: `{a <- not_b}` has the only stable model `{a, not_b}` while `{b <- not_a}` has only `{b, not_a}` as stable model. Equally, {`neg_a`, `neg_b`}⊗{`a <- neg_b`} has only `{a, neg_b}` and {`neg_a`, `neg_b`}⊗{`b <- neg_a`} has only `{b, neg_a}` as 2-models. This can be checked directly along the lines of the example in Section 4.

Remark 1. The issue of contrapositive in logic programming is widely discussed, see, for instance, [Pearce, 1997, § 7.2]. We give here an example of *pedestrian lights* which should illustrate how it 'works' in our case.

Pedestrian lights are characterized by the two atoms `red` and `green`. Whether one should cross the street is determined by two rules:

```
go :- green
neg_go :- red
```

Now, the idea of default negation implies that, as long as I don't see the lights, neither `green` nor `red` should be assumed but rather the contrary. Thus, in our initial programme $\mathcal{E}_0$ we will have the two atoms: `neg_red` and `neg_green`. In this way, one cannot conclude whether one could cross the street or not. In this context, one can safely (in a literal sense!) assume the following rule:

```
red :- neg_green
```

It expresses that, as long as I don't have *positive* information that the lights are green, I 'better' assume that it is red (and, with the rules above, I conclude not to go). In contrast, the contrapositive of the rule (i.e., `green :- neg_red`) should not be assume (just imagine the lights are broken).

We added this example, as it illustrates two aspects of default negation, as they becomes visible in our reading as explicit negation plus update: First, the initial 'agnostic' state gives (total) preference to negative information; this alone shoud not be used to conclude positive actions (here: to cross the street); if there should be consequences concluded, they have to be given explicitely (here: to consider the lights being red as long as one doesn't have positive information about green) — but these rules may be *intensional* as they should not imply all of their consequences in classical logic. Secondly, updates allows to overwrite default assumptions, for instance, when one is seeing the green light.

One may ask how our approach works for *double negation*. Our syntax does not allow to iterate `neg`, and one would, first, have to change the language, introducing negation as an operator (rather than a prefix). The concept of 2-models would profit from an annulation of double negation. If this is not the case, a new fact `neg neg a` would have to rule out `neg a` in a model (that's the minimum we would expect from a 'negation'), but would not give support for `a`, thus 'distroying' the 2-valuedness of a model. In consequence, our set-up for explicit dynamic logic programs would fail, as updates could result in the 'destruction' of all models. The question how to deal with double negation in our framework, thus, is subject to further investigation.

7.9. Disjunctive logic programming

The question whether we can extend our framework to *disjunctive logic programming* [Minker, 1994] is even more challenging. *Extended* disjunctive

logic programming is insofar out of reach, as we do not combine explicit and default negation in the same framework (see § 7.3.). For normal disjunctive logic programs, we would expect that our translation should work conceptionally; however, in this case, where 'disjunctions' (given, for example, as lists of atoms) can occur in the heads of rules, one would have, first, to redefine the update operation, as the notion of *conflicting* rules (Definition 3 and 6) is not any longer straightforward and as, in addition, the *Reject* operation (Definition 4) needs to be significantly refined. This applies, of course, also to *generalized disjunctive logic programs* [Lifschitz, 1996] where default negation may occur in the disjunction of a head of a rule and *general disjunctive programs* [Shen and Eiter, 2019], which take even into consideration arbitrary first-order formulas. It is to expect that one can go along the increasing complexity of the heads in disjunctive logic programming to introduce corresponding update frameworks, but to develop such frameworks is future work.

8. Coda

In the present paper, we gave a reading of default negation as explicit negation with update, which is a form to formalize the '*commonsense law of inertia*, which is the principle that things do not change unless they are made to' [Przymusinski and Turner, 1997, p. 126].

Form the perspective of Computer Science, our approach may give rise to more investigations of update phenomena in *answer set programming* [Slota and Leite, 2010; Slota and Leite, 2014] and can be linked to *action languages* (stemming from [Gelfond and Lifschitz, 1998]). Also, a more profound comparison with other semantical approaches to Answer Set Programming, as *equilibrium logic* and 'Here-and-There'-models [Pearce, 1997; Odintsov and Pearce, 2005; Pearce, 2006] and the related *Strong-Equivalence*-models [Turner, 2003]. This is or particular interest in view of the skeptical evaluation of these accounts for updates in [Slota and Leite, 2014]. This will be investigated in the future.

The purpose of this paper could be characterized as more philosophical: the reading of default negation as explicit negation plus update illustrates how the non-monotonic nature of default negation can be located in the update step. Methodologically, it allows to *modularize* semantic questions, concerning the default assumptions and updates. In this way, we hope to contribute to the analysis of non-monotonicity, not only in logic programming, but in philosophical logic in general.

Acknowledgements. This work is partially supported by the Udo Keller Foundation and by the Portuguese Science Foundation, FCT, through the project

UID/MAT/00297/2020 (Centro de Matemática e Aplicações). The author is grateful to an anonymous referee for helpful comments.

References

Alferes et al., 2000 – Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H. and Przymusinski, T.C. "Dynamic Updates of Non-Monotonic Knowledge Bases", *The Journal of Logic Programming*, 2000, Vol. 45, No. 1–2, pp. 43–70.

Banti et al., 2004 – Banti, F., Alferes, J.J. and Brogi, A. "Well Founded Semantics for Logic Program Updates", in: *Advances in Artificial Intelligence – IBERAMIA 2004*, Volume 3315 of *Lecture Notes in Computer Science*, ed. by Chr. LemaÎtre, C.A. Reyes, and J.A. González. Springer, 2004, pp. 397–407.

Bidoit and Froidevaux, 1991 – Bidoit, N., and Froidevaux, C. "General logic databases and programs: Default logic semantics and stratification", *Information and Computation*, 1991, Vol. 91, No. 1, pp. 15–54.

Buccafurri et al., 1999 – Buccafurri, F., Faber, W. and Leone, N. "Disjunctive Logic Programs with Inheritance", in: *Proceedings of the 1999 International Conference on Logic Programming (ICLP-99)*, ed. by D. De Schreye. Cambridge: MIT Press, 1999, pp. 79–93.

Clark, 1978 – Clark, K.L. "Negation as failure", in: *Logic and Data Bases*, ed. by H. Gallaire and J. Minker. Plenum, 1978, pp. 293–322.

Damásio and Pereira, 1996 – Damásio, C.V., and Pereira, L.M. "Default Negation in the heads: Why not?", in: *Extensions of Logic Programming, ELP'96*, ed. by R. Dyckhoff, H. Herre, and P. Schroeder-Heister, Volume 1050 of *Lecture Notes in Artificial Intelligence*. Springer, 1996, pp. 103–117.

Eiter et al., 2002 – Eiter, Th., Fink, M., Sabbatini, G. and Tompits, H. "On properties of update sequences based on causal rejection", *Theory and Practice of Logic Programming*, 2002, Vol. 2, No. 6, pp. 711–767.

Eshghi and Kowalski, 1989 – Eshghi, K., and Kowalski, R.A. "Abduction compared with negation as failure", in: *Proc. 6th International Conference on Logic Programming*, ed. by G. Levi and M. Martelli. MIT Press, 1989, pp. 234–255.

Gelder, et al., 1991 – Gelder, A. Van, Ross, K.A. and Schlipf, J.S. "The well-founded semantics for general logic programs", *Journal of the ACM*, 1991, Vol. 38, No. 3, pp. 620–650.

Gelfond and Lifschitz, 1988 – Gelfond, M., and Lifschitz, V. "The Stable Model Semantics for Logic Programming", in: *5th International Conference on Logic Programming*, ed. by R. Kowalski and K. A. Bowen. MIT Press, 1988, pp. 1070–1080.

Gelfond and Lifschitz, 1998 – Gelfond, M., and Lifschitz, V. "Action Languages", *Electronic Transactions on Artificial Intelligence*, 1998, Vol. 2, pp. 193–210.

Gelfond and Lifschitz, 1991 – Gelfond, M., and Lifschitz, V. "Classical negation in logic programs and disjunctive databases", *New Generation Computing*, 1991, Vol. 9, No. 3–4, pp. 365–385.

Inoue and Sakama, 1998 – Inoue, K., and Sakama, C. "Negation as failure in the head", *Journal of Logic Programming*, 1998, Vol. 35, pp. 39–78.

Jonker, 1994 – Jonker, C. "Constraints and Negations in Logic Programming", Ph.D. diss., Department of Philosophy, Utrecht University, 1994.

Kakas et al., 1998 – Kakas, A.C., Kowalski, R.A. and Toni, F. "The Role of Abduction", in: *Handbook of Logic in Artificial Intelligence and Logic Programming*, ed. by D.M. Gabbay, C.J. Hogger, and J.A. Robinson, Volume 5: Logic Programming. Oxford, 1998, pp. 235–324.

Leite, 1997 – Leite, J.A. "Logic Program Updates", Master's thesis, Dept. de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 1997.

Leite, 2003 – Leite, J.A. *Evolving Knowledge Bases*. Volume 81 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2003.

Leite, 2004 – Leite, J.A. "On Some Differences Between Semantics of Logic Program Updates", *Advances in Artificial Intelligence – IBERAMIA 2004*, Volume 3315 of *Lecture Notes in Computer Science*, ed. by Chr. LemaÎtre, C.A. Reyes, and J.A. González. Springer, 2004, pp. 375–385.

Leite and Pereira, 1998a – Leite, J.A., and Pereira, L.M. "Generalizing updates: from models to programs", *LPKR'97: ILPS'97 workshop on Logic Programming and Knowledge Representation*. Springer, 1998. pp. 224–246.

Leite and Pereira, 1998b – Leite, J.A., and Pereira, L.M. "Iterated Logic Program Updates", in: *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming (JICSLP-98)*, ed. by J. Jaffar. Cambridge: MIT Press, 1998, pp. 265–278.

Lifschitz, 1996 – Lifschitz, V. "Foundations of logic programming", in: *Principles of Knowledge Representation*, ed. by G. Brewka. Stanford, California: CSLI, 1996, pp. 69–128.

Lifschitz et al., 2001 – Lifschitz, V., Pearce, D. and Valverde, A. "Strongly equivalent logic programs", *ACM Transactions on Computational Logic*, 2001, Vol. 2, pp. 526–541.

Lifschitz and Woo, 1992 – Lifschitz, V., and Woo, T. "Answer sets in general nonmonotonic reasoning (Preliminary report)", in: *Principles of Knowledge Representation and Reasoning (KR92)*, ed. by B. Nebel, C. Rich, and W. Swartout. Morgan-Kaufmann, 1992, pp. 603–614.

Minker, 1994 – Minker, J. "Overview of disjunctive logic progamming", *Annals of Mathematics and Artificial Intelligence*, 1994, Vol. 12, pp. 1–24.

Odintsov and Pearce, 2005 – Odintsov, S., and Pearce, D. "Routley Semantics for Answer Sets", in: *Logic Programming and Nonmonotonic Reasoning*, ed. by Ch. Baral, G. Greco, N. Leone, and G. Terracina, Volume 3662 of *Lecture Notes in Artificial Intelligence*. Springer, 2005, pp. 343–355.

Pearce, 1997 – Pearce, D. "A new logical characterization of stable models and answer sets", in: *Non-Monotonic Extensions of Logic Programming*, ed. by J. Dix, L.M. Pereira, and T. Przymusinski, Volume 1216 of *Lecture Notes in Computer Science*. Springer, 1997, pp. 57–70.

Pearce, 1999 – Pearce, D. "From here to there: Stable negation in logic programming", in: *What is Negation?*, ed. by D. Gabbay and H. Wansing. Kluwer, 1999, pp. 161–181.

Pearce, 2006 – Pearce, D. "Equilibrium logic", *Annals of Mathematics and Artificial Intelligence*, 2006, Vol. 47, pp. 3–41.

Poole, 1994 – Poole, D. "Default Logic", in: *Handbook of Logic in Artificial Intelligence and Logic Programming*, ed. by Dov Gabbay, C.J. Hogger, and J.A. Robinson, Vol. 3. Oxford, 1994, pp. 189–215.

Przymusinski and Turner, 1997 – Przymusinski, T.C., and Turner, H. "Update by means of inference rules", *The Journal of Logic Programming*, 1997, Vol. 20, pp. 125–143.

Reiter, 1978 – Reiter, R. "On closed world data bases", in: *Logic and Data Bases*, ed. by H. Gallaire and J. Minker. Plenum, 1978, pp. 55–76.

Reiter, 1980 – Reiter, R. "A Logic for Default-Reasoning", *Artificial Intelligence*, 1980, Vol. 13, pp. 81–132.

Sakama and Inoue, 1999 – Sakama, C., and Inoue, K. "Updating Extended Logic Programs through Abduction", *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-99)*, Volume 1730 of *LNAI*, ed. by M. Gelfond, N. Leone, and G. Pfeifer. Berlin: Springer, 1999, pp. 147–161.

Shen and Eiter, 2019 – Shen, Y.-D., and Eiter, T. "Determining inference semantics for disjunctive logic programs", *Artificial Intelligence*, 2019, Vol. 277, pp. 103–165.

Shepherdson, 1998 – Shepherdson, J.C. "Negation as Failure, Completion and Stratification", in: *Handbook of Logic in Artificial Intelligence and Logic Programming*, ed. by D.M. Gabbay, C.J. Hogger, and J.A. Robinson, Vol. 5: Logic Programming. Oxford, 1998, pp. 355–419.

Slota et al., 2014 – Slota, M., Baláž, M., and Leite, J. "Supporting Strong and Default Negation in Answer-Set Program Updates", *Advances in Artificial Intelligence – IBERAMIA 2014*, Volume 8864 of *Lecture Notes in Computer Science*, ed. by A. Bazzan and K. Pichara. Springer, 2014, pp. 41–53.

Slota and Leite, 2010 – Slota, M., and Leite, J. "On Semantic Update Operators for Answer-Set Programs", in: *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. IOS Press, 2010, pp. 957–962.

Slota and Leite, 2014 – Slota, M., and Leite, J. "The Rise and Fall of Semantic Rule Updates Based on SE-Models", *Theory and Practice of Logic Programming*, 2014, Vol. 14, pp. 869–907.

Turner, 2003 – Turner, H. "Strong equivalence made easy: nested expressions and weight constraints", *Theory and Practice of Logic Programming*, 2003, Vol. 3, pp. 609–622.

Zhang and Foo, 1998 – Zhang, Y., and Foo, N.Y. "Updating Logic Programs", in: *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, ed. by Henri Prade. Chichester: John Wiley & Sons, 1998, pp. 403–407.