Theoretical Computer Science 78 (1991) 85-112
North-Holland

Negation by default and unstratifiable logic programs

Nicole Bidoit
U.A. 410 du CNRS, LRI, 91405 Orsay Cedex, France

Christine Froidevaux
Université Paris XI, LRI, 91405 Orsay Cedex, France

Abstract

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

The default approach to the theory of logic programs (and deductive databases) is based on the interpretation of negation by default rules. Default logic is a well-suited formalism to express the Closed World Assumption and to define the declarative semantics of stratifiable logic programs. The case of disjunctive consequences in rules is treated. General logic programs may not have a meaning with respect to default semantics. The contribution of the paper is to exhibit an interesting class of programs having a default semantics, called effectively stratifiable programs. This time, disjunctive consequences are not considered. Effective stratification is a weaker constraint than stratification, local stratification and weak stratification. Besides enlarging the class of stratifiable logic programs, the paper contributes to provide a constructive definition of well-founded models of logic programs. The class of effectively stratifiable logic programs matches the class of programs having a total well-founded model and in general, the default semantics extends the well-founded semantics.

1. Introduction

A general logic program is a set of rules that have both positive and negative premises. Recently, much work has been done in order to develop a semantics for general logic programs. The core of the problem is to assign negation as failure a declarative, model-theoretic meaning which justifies its use [24]. Two approaches have been followed: the "program completion" approach and the "canonical model" approach.

Following the "program completion" approach [10, 22], given a general logic program P, a theory called the completion of P is associated with P and the negative literals that are theorems of the completion of P are the only negative literals

derivable by SLD plus Negation as Failure from the initial program. This approach has been extensively studied in [2, 22, 31, 32]. One of the weaknesses of this approach lies in its "unnatural" behavior for many simple examples [35].

Following the "canonical approach", one of the models of the general program P is elected to be the model that captures the "intended meaning" of the program P. Usually, the "canonical model" of P is selected among the minimal models of P. Clearly, the choice of the canonical model of general logic programs highly relies on the common sense of the programmer or/and the user of the program. Several methods [11, 1, 7, 8, 27, 23, 26, 34, 3, 4, 18, 35, 19] have emerged during the last years as better or worse theoretical tools to define the declarative semantics of logic programs with negation. All these approaches have in common the following problem: a general logic program may not have a canonical model. Besides this, for some of these approaches like [7, 8, 27], it is unclear whether the canonical model really reflects the intended meaning of the program. The computational aspect of the problem should also be taken into account.

Classes of programs that have a canonical model, i.e. a clear intended meaning, have been exhibited. Stratifiable programs were first introduced in [11] and further studied by [1, 34, 27] and others. Stratification restricts the "definition" of predicates to be independent of their complement or negation. Stratification can be directly and easily determined from the syntax of the logic program. Checking stratification can be reduced to checking the existence of cycles in the graph associated with the program [1]. The class of locally stratifiable programs [27] includes stratifiable programs. Local stratification restricts the definition of atoms of the Herbrand base to be independent of their negation. One needs to consider the instantiation of the logic program in order to check whether it is locally stratifiable. Moreover, although local stratification is of particular interest because it allows one to consider logic programs with function symbols, checking local stratification for logic programs with function symbols is computationally unreasonable.

More recently [5, 29, 35], local stratification has been shown to characterize a too restrictive class of "well-behaved" programs. As a matter of fact, some very simple and useful logic programs, without function symbols, are exhibited [5, 29, 35] that are neither stratifiable nor locally stratifiable but make good sense or in other words have a clear intended meaning.

The aim of this paper is to define a wider class of legal or "well-behaved" logic programs. We are thus seeking for a less restrictive constraint than local stratification. This condition, called effective stratification, should obviously ensure the existence of a canonical model. Intuitively, effective stratification is a refinement of (local) stratification in the following sense: in the instantiation of a logic program P, irrelevant rules or definitions are discarded and the initial program P is said to be effectively stratifiable if the remaining "effective" rules of its instantiation constitute a stratifiable logic program. Effective stratification has been introduced in [6]. Given an effectively stratifiable program P (that is a program which can be transformed by a "sound process" described in the paper into an equivalent stratifiable program),

modifying the program by updating the set of facts in P may lead to a program which is not effectively stratifiable. In that respect, effective stratification is data dependent.

For logic programs without function symbols, testing effective stratification and computing canonical model of effectively stratifiable programs can be performed in polynomial time.

In the paper, the intended meaning of general logic programs is defined using the default logic formalism. Default logic has been introduced by [30] in order to formalize nonmonotonic reasoning. Previous work [7, 8, 3, 4, 5, 6] has proved that default logic is a well-suited formalism for expressing the Closed World Assumption and for defining the declarative semantics of general logic programs including disjunction in the consequence of rules.

In this framework, the declarative analog of the procedural Negation as Failure interpretation of negation is negation by default. A general logic program is viewed as a particular default theory [30] and its semantics is captured by the extensions of that default theory. In [30] extensions of a default theory are defined as fixed points of a non-monotonic operator. In fact, as shown in [3, 4], extensions of the default theory associated with a logic program satisfy a number of good properties. First, while in general, extensions of a default theory may not be maximally consistent, extensions of logic programs are always maximally consistent. Thus an extension of a logic program can be identified with an interpretation. This interpretation is a model of the logic program and is called a default model in the following. Moreover, a default model of a logic program P is a minimal model of P in the sense of [9]. Finally, it is in our opinion very important to emphasize that a default model of a logic program is supported [1], or equivalently, justified [8]. This last property is a very natural property, which one can expect to be satisfied by the canonical model intended to capture the meaning of a logic program [1, 7, 4, 35]. Note, for instance, that perfect models [28] of a logic program P are minimal for P but not always supported for P.

As for other approaches, a logic program may not have a default model or may have more than one default model. In the first case, we say that the program is inconsistent and in the second, we say it is ambiguous. Obviously, our interest focuses on programs that are consistent and unambiguous or in other words that have a unique default model. In [3, 4], stratifiable logic programs as well as locally stratifiable programs are shown to be consistent and unambiguous. It is also shown that for (locally) stratifiable programs, the default logic approach is equivalent to three other major approaches (known to be equivalent for stratifiable programs [23, 27]): the iterative fixed point semantics [1], the perfect model semantics [27] and the circumscriptive semantics [23].

The paper shows that effectively stratifiable logic programs have a unique default model. Effective stratification is a sufficient condition for consistency and unambiguity of logic programs. However, some logic programs that are not effectively stratifiable but have a unique default model can be exhibited. We would like to

insist here on the fact that effective stratification characterizes a subclass of the programs having a unique default model and that failure for a logic program to satisfy the effective stratification constraint does not entail that this program makes no sense (with respect to default semantics).

Effective stratification is a generalization of sup-stratification [5] and weak stratification [29]. Because the perfect model semantics reveals itself to be too weak a semantics for capturing the meaning of some not locally stratifiable but "well-behaved" programs, it has been extended in [29] where weakly perfect models are introduced. It is important to note that effectively stratifiable programs may not have a weakly perfect model. From a purely semantical point of view, while some logic programs have a unique default model but no weakly perfect model, logic programs consistent with respect to the weakly perfect model semantics may be inconsistent with respect to the default logic approach. Just as perfect models, weakly perfect models may not be supported. The notion of (unique) stable model introduced in [19] as a new semantics for logic programs is identical to the previously introduced notion of default model [3, 4]. Finally, it is interesting to see that a logic program has a well-founded model [35] iff that program is effectively stratifiable while, in general, the notion of default model extends the notion of well-founded model.

The paper is organized as follows. Section 2 motivates the need for relaxing local stratification and the choice of default logic as a theoretical tool for defining the declarative semantics of logic programs. Section 3 simply recalls some basic concepts and notations used throughout the paper. Section 4 formally introduces the constraint of effective stratification and investigates its properties. The main result of Section 4 states that effective stratification entails the existence and uniqueness of the default model of a logic program. Finally, Section 5 includes a comparative discussion. Weak stratification [29] versus effective stratification, weakly perfect model [29] versus default model, stable model [19] versus default model and well-founded model [35] versus default model are successively examined.

2. Motivation

First we illustrate the limitations of stratification and local stratification. Intuitively, stratification and local stratification are two constraints that disallow the intentional definition of relations or predicates to make use of their complement or negation. *Local stratification* is a refinement of *stratification* in the following sense. On the one hand, roughly speaking, stratification is a constraint on the predicate symbols of the first-order language used to specify a program P; stratification forbids a predicate A to be "defined" using its negation in a program P. On the other hand, intuitively, local stratification is a constraint on the *atoms of the Herbrand Base* HB_L of the language L used to define a program P; local stratification disallows, *in the instantiation* of a program P, an element $A(a)$ of HB_L to be "defined" using its

negation $\neg A(a)$. The following examples illustrate these informal definitions.

Example 2.1. The following program due to Lifschitz [27] intends to define even numbers.

$$P1 = \{\text{EVEN}(0), \neg\text{EVEN}(x) \rightarrow \text{EVEN}(\text{successor}(x))\}.$$

The instantiation of $P1$ is given by:

$$I\_P1 = \{\text{EVEN}(0)\} \cup \{\neg\text{EVEN}(\text{successor}^i(0)) \rightarrow \text{EVEN}(\text{successor}^{i+1}(0)) \mid i \geqslant 0\}.$$

Since the predicate EVEN occurs both in the right-hand side of the second rule of $P1$ and negatively in the left-hand side of this rule, the logic program $P1$ is not stratifiable. However, it is easy to check that $P1$ is locally stratifiable because in the instantiation I_P1 of the logic program $P1$, there is no definition of EVEN(successor^i(0)) that depends on $\neg$EVEN(successor^i(0)).

We now give an example of a program that is not locally stratifiable but has a "well-defined semantics". Several examples of the same kind can be found in [35, 29, 19].

Example 2.2. Our second example is in fact very similar to Lifschitz's example and describes a genealogical tree in which each individual inherits a property E every other generation.

$$P2 = \{\text{Father}(a, b), \text{Father}(b, c), E(a), \text{Father}(x, y) \wedge \neg E(x) \rightarrow E(y)\}.$$

The instantiation of $P2$ is given by

$$\begin{aligned}
I_P2 = \{&\text{Father}(a, a) \wedge \neg E(a) \rightarrow E(a) && (1),\\
&\text{Father}(a, b) \wedge \neg E(a) \rightarrow E(b) && (2),\\
&\text{Father}(a, c) \wedge \neg E(a) \rightarrow E(c) && (3),\\
&\text{Father}(b, a) \wedge \neg E(b) \rightarrow E(a) && (4),\\
&\text{Father}(b, b) \wedge \neg E(b) \rightarrow E(b) && (5),\\
&\text{Father}(b, c) \wedge \neg E(b) \rightarrow E(c) && (6)\\
&\text{Father}(c, a) \wedge \neg E(c) \rightarrow E(a) && (7),\\
&\text{Father}(c, b) \wedge \neg E(c) \rightarrow E(b) && (8),\\
&\text{Father}(c, c) \wedge \neg E(c) \rightarrow E(c) && (9),\\
&E(a) && (10),\\
&\text{Father}(a, b) && (11),\\
&\text{Father}(b, c) && (12)\}.
\end{aligned}$$

The logic program $P2$ is neither stratifiable nor locally stratifiable. The instantiation I_P2 of the program $P2$ contains, for instance, the rule Father$(a, a) \wedge \neg E(a) \rightarrow E(a)$; this rule defines the atom $E(a)$ using its negation $\neg E(a)$.

Note here that the first-order language used to specify the program $P2$ does not include function symbols.

A strong correspondence between the program $P1$ and the program $P2$ is easy to notice. Because the two programs $P1$ and $P2$ are similar and because the first one, program $P1$, is considered to be "legal" and has a well-defined semantics, one naturally expects the program $P2$ to belong to the class of "legal" programs and to have a well-defined semantics. Note that $P2$ is, to some extent simpler than $P1$ because it does not contain function symbols.

We already have shown that the program $P2$ is not legal in the sense that $P2$ is not locally stratifiable. Concerning the declarative semantics associated with the program $P2$, it is quite interesting to note here that, following the perfect model approach [27], $P2$ is viewed as an inconsistent or meaningless program: $P2$ has no perfect model. The inconsistency of $P2$ with respect to the perfect model approach is rather unfortunate. Example 2.2 reveals the need to relax the constraint of (local) stratification and also the limitation of the perfect model approach. In order to associate a semantics with $P2$, one needs not only to relax the constraint of local stratification but also to consider a new semantics. In [29], both an extension of the class of (locally) stratifiable programs and a "natural" extension of perfect model are proposed. Weak stratification and weakly perfect model will be discussed in Section 5.

On the other hand, following the default logic approach [3, 4], the logic program $P2$ is consistent and unambiguous. Indeed, $P2$ has a unique default model. The default model of $P2$ is $\{\text{Father}(a, b), \text{Father}(b, c), E(a), E(c)\}$. It perfectly matches our intuition of the intended semantics of $P2$ and moreover corresponds to the Prolog or SLDNF procedural interpretation of $P2$.

Let us now use Example 2.2 in order to intuitively introduce the constraint of *effective* stratification. Clearly, considering the instantiation I_P2 of the program $P2$ as a refinement of $P2$ is not very natural. As a matter of fact, in I_P2, rules such as (1), (3), (4), (5), (7), (8) and (9) are irrelevant. From a procedural point of view, they will never be activated. These rules are not *effective* rules. For instance, rule (1) is irrelevant because one knows that "a" is not his own Father.

Thus, our goal is, given a program P, to find a "good" refinement of P that exclusively contains effective rules. For our example, the good refinement of $P2$ is the set of rules $\text{Ref\_}P2 = \{(2), (6), (10), (11), (12)\}$. Now, given a "good" refinement $\text{Ref\_}P$, the initial program P is "legal" or satisfies the effective stratification constraint if no atom $A(a)$ of HB_L is "defined" in $\text{Ref\_}P$ using its negation $\neg A(a)$, or in a simpler way, if the program $\text{Ref\_}P$ is stratifiable. The logic program $P2$ is effectively stratifiable.

3. Preliminaries

In this section, we present and review some basic concepts of logic [12, 16, 17], default logic [30] and logic programming [2, 22]. We also introduce the notations used throughout the paper.

From the preceding section, it should be clear that the condition of effective stratification will be formally defined for a first-order logic program P by requiring some property to be satisfied by (a subset of) the Herbrand instantiation of P. The instantiation of a program can be viewed as a propositional program. Thus, in order to simplify the further discussion, we restrict our attention to propositional programs (in other words, we directly consider instantiations of programs). In the following, we denote by Prop a finite set of propositions.

We deliberately choose not to consider infinite sets of propositions (that intuitively would serve to take care of instantiation of first-order logic programs with function symbols). A discussion on the infinite case is nevertheless carried on at the end of Section 4.

In order to simplify the presentation, we choose here to stick as much as possible with the classical notations of logic programming. In particular, from a syntactical point of view, while using default logic, we write a logic program as a set of rules rather than as a set of default rules. For the interested reader, [3, 4] provide a detailed presentation of how logic programs and logical databases can be specified by default theories.

Thus, in the following, a logic program P is a set of rules; a rule r is a formula of the form:

$$A_1 \wedge \cdots \wedge A_p \wedge \neg B_1 \wedge \cdots \wedge \neg B_q \rightarrow C,$$

where $p \geqslant 0$, $q \geqslant 0$ and $A_1, \ldots, A_p, B_1, \ldots, B_q, C$ are propositions. We denote by $\text{Prem}(r)$ the set $\{A_1, \ldots, A_p, \neg B_1, \ldots, \neg B_q\}$ of premises of r, and by $\text{Cons}(r)$ the consequent C of r.

A *literal* is a proposition or the negation of a proposition. In the first case, it is called a *positive literal*, in the second case, it is called a *negative literal*.

Given a set of literals Lit, we denote by Lit^+ (resp. Lit^-) the subset of positive (resp. negative) literals in Lit. Thus, for instance $\text{Prem}(r)^+$ (resp. $\text{Prem}(r)^-$) denotes the set of positive (resp. negative) premises of r.

A rule r is called a *positive rule* if $\text{Prem}(r)^- = \emptyset$.

Given a logic program P, we denote by P^+ the subset of positive rules in P and by P^- the subset of negative rules in P.

$$P^+ = \{r \mid r \in P \text{ and } \text{Prem}(r)^- = \emptyset\} \quad \text{and} \quad P^- = P - P^+.$$

For a set Lit of literals, we denote by $\neg.\text{Lit}$ the set of literals obtained by taking the negation of each literal in Lit, that is $\neg.\text{Lit} = \{\neg L \mid L \in \text{Lit}\}$ (of course, we make the convention that $\neg\neg L$ is L).

The declarative semantics of a logic program P is defined in [3, 4] by means of the set of *extensions* of the default theory associated with P. These extensions are defined in [30] as fixed points of a non-monotonic operator. In the following, for sake of simplicity, we will use an equivalent definition for default models of P. It is straightforward to prove that the definition below is equivalent to the definition of a default model as the model of an extension.

We make the convention to represent *interpretations* by maximally consistent sets of literals. Given a set of propositions Prop, an interpretation I over Prop is a set of literals over Prop such that

(1) if L is in I then $\neg L$ is not in I, and

(2) if A is in Prop then either A or $\neg A$ is in I.

Intuitively a default model M of P is an interpretation which is able to reproduce itself from the negative literals in M and from the rules of P.

In a slightly different way than in [2], an *operator* E_P is associated with a logic program P as follows:

$$E_P(S) = S \cup \{\text{Cons}(r) \mid r \in P \text{ and } \forall L \in \text{Prem}(r),\ L \in S\},$$

where S is a set of literals over Prop. An interpretation M is a *default model* of the program P iff $M = E_P(M)$, i.e. iff M is a fixpoint of E_P.

Another way to express that an interpretation M is a default model of P follows. Consider the sequence $(M_i)_{(i \geqslant 0)}$ where

$$M_0 = M^- \quad \text{and} \quad M_{i+1} = E_P(M_i), \quad \text{for } i \geqslant 0.$$

Then (note that the sequence $(M_i)_{(i \geqslant 0)}$ is an increasing sequence), an interpretation M is a *default model* of the program P iff $M = \bigcup_{i \geqslant 0} M_i$.

A default model of a logic program P is a *minimal model* of the program P considered as a set of clauses. By minimal [9], we mean that the set of positive literals is minimized.

A default model of a logic program P is a *supported model* of P. This property is linked to the fact that default semantics is defined as a fixpoint semantics. The notion of supported model has been introduced in [1] and simultaneously in [7, 8] where it is called a *causal* or *justified* model. Roughly speaking, given a model M for a logic program P, a *proposition* A true in M is *supported* by P iff there exists a definition r of A in P (that is a rule with consequent A) such that each premise of r is satisfied by M. A model M for P is a supported model of P iff each true proposition A in M is supported by P.

Note that the converse does not hold: a minimal and supported model of P may not be a default model of P. For instance, $P = \{A \to A, \neg A \to A\}$ has a minimal and supported model $M = \{A\}$ but no default model because M cannot produce itself starting from $M^- = \emptyset$. In fact A is artificially supported by the rule $A \to A$. Default model semantics does not have this drawback.

A logic program may not have a default model. It is then called *inconsistent*. For instance, the program $\{\neg A \to A\}$ has no default model. On the other hand, a logic program may have more than one default model. It is then called *ambiguous*. For instance the logic program $\{\neg A \to B, \neg B \to A\}$ has two default models, namely $\{A, \neg B\}$ and $\{\neg A, B\}$.

Stratification forbids recursion involving negation. Formally, a program P is *stratifiable* iff there is a sequence $S = \text{Prop}_1 \cdots \text{Prop}_s$, such that S is a partition of

Prop satisfying:

$$\forall r \in P, \quad \mathrm{Cons}(r) \in \mathrm{Prop}_i \quad \text{implies}$$

$$\forall A \in \mathrm{Prem}(r)^+, \exists k \leqslant i \,|\, A \in \mathrm{Prop}_k \text{ and}$$

$$\forall \neg A \in \mathrm{Prem}(r)^-, \exists k < i \,|\, A \in \mathrm{Prop}_k.$$

Apt et al. [1] show that checking stratification can be reduced to checking the existence of cycles in the graph associated with the program.

In [3, 4] the consistency and the unambiguity of (locally) stratifiable programs is proved by establishing the existence and the uniqueness of extensions for (locally) stratifiable default theories associated with these programs.

4. Effective stratification

The contents of this section were outlined in Section 2. We directly proceed to the formal presentation. In order to define the constraint of effective stratification, given a program P, we utilize some simple properties of propositions for the program P.

Intuitively, a proposition A is defined in a logic program P if A is a logical consequence of the positive part of P, i.e. $P^+ \vdash A$. Now, a proposition A is potentially defined in a logic program P if A can be inferred from the rules of P while ignoring the negative premises of these rules if any.

Definition 4.1. Let P be a logic program over Prop. Then:

(a) *Defined* propositions in P are recursively defined as follows:

(1) if A is in P then A is defined in P,

(2) if $\exists r \in P^+ \,|\, \mathrm{Cons}(r) = A$ and $\forall B \in \mathrm{Prem}(r)$, B is defined in P, then A is defined in P, and

(3) all defined propositions in P are generated by applying the two above rules.

The set of defined propositions in P is denoted by $\mathrm{Def}(P)$.

Formally we have $\mathrm{Def}(P) = \bigcup_{i=1,\ldots,\infty} E'_P(\emptyset)$.

(b) *Potentially defined* propositions in P are recursively defined as follows:

(1) if A is in P then A is potentially defined in P,

(2) if $\exists r \in P \,|\, \mathrm{Cons}(r) = A$ and $\forall B \in \mathrm{Prem}(r)^+$, B is potentially defined in P, then A is potentially defined in P, and

(3) all potentially defined propositions in P are generated by applying the above rules.

The set of potentially defined propositions in P is denoted by $\mathrm{Potdef}(P)$.

Formally we have: $\mathrm{Potdef}(P) = \bigcup_{i=1,..,\infty} \mathrm{Potd}'_P$, where

$$\mathrm{Potd}^1_P = \mathrm{Def}(P) \text{ and for } i \geqslant 1$$

$$\mathrm{Potd}^{i+1}_P = \mathrm{Potd}^i_P \cup \{\mathrm{Cons}(r) \,|\, \forall B \in \mathrm{Prem}(r)^+, B \in \mathrm{Potd}^i_P\}.$$

(c) A proposition A is *undefined* in P iff A is not potentially defined in P. The set of undefined propositions in P is denoted by Undef(P).

$$\text{Undef}(P) = \text{Prop} - \text{Potdef}(P).$$

Obviously, since Def(P) is included in Potdef(P), the sets Def(P) and Undef(P) are disjoint.

Before presenting examples to illustrate the previous definition, one should note that the notions of defined and undefined propositions proposed above generalize the notions of defined and undefined propositions introduced in [5].

Let us now illustrate the above definition.

Example 4.2. Let Prop $= \{A, B, C, D, E, F, G\}$ and consider the following logic program P over Prop:

$$P = \{A,\ A \rightarrow G,\ \neg A \rightarrow D,\ B \wedge \neg C \rightarrow D,\ A \wedge \neg B \wedge \neg D \rightarrow C,\ E \rightarrow F,\ F \rightarrow E\}.$$

Note that P is not stratifiable.

The proposition A is defined in P because A is in P; hence A is potentially defined. The proposition G is defined in P because $A \rightarrow G$ is in P and A is defined in P.

The proposition C (respectively D) is potentially defined in P because the unique positive premise A of the rule $A \wedge \neg B \wedge \neg D \rightarrow C$ that defines C in P is potentially defined in P (respectively because the rule $\neg A \rightarrow D$, part of the definition of D in P, has no positive premise and thus a fortiori each positive premise of this rule is potentially defined in P).

Finally the proposition B (respectively E and F) is undefined in P because there is no definition of B in P (respectively because E and F are not potentially defined in P).

Example 4.3. Let Prop $= \{A, B, C\}$ and consider the following logic program P over Prop:

$$P = \{A \rightarrow B,\ B \rightarrow A,\ \neg A \rightarrow C,\ \neg C \wedge B \rightarrow A\}.$$

Note that P is not stratifiable.

The set of defined propositions in P is empty. The proposition C is potentially defined in P because the rule $\neg A \rightarrow C$ that defines C in P has no positive premises. The propositions A and B are undefined in P.

The following result shows that, given an interpretation M, the sequence $(M_i)_{(i \geq 0)}$, used to check whether M is a default model of P, contains all the defined propositions in P and also that all propositions in that sequence are, at least, potentially defined in P.

Lemma 4.4. *Let P be a logic program over* Prop, *let A be in* Prop *and let M be an interpretation. Let us consider the sequence $(M_i)_{i\geqslant 0}$ where $M_0 = M^-$ and $M_{i+1} = E_P(M_i)$ for $i \geqslant 0$. Then*:

(i) *if $A \in \bigcup_{i=0,\ldots,\infty} M_i$ then A is potentially defined in P,*

(ii) *if A is defined in P then $A \in \bigcup_{i=0,\ldots,\infty} M_i$.*

Proof. (i) Let A be a proposition in $\bigcup_{i=0,\ldots,\infty} M_i$. We show by induction on i that $A \in \text{Potdef}(P)$. Since A is a proposition, A cannot belong to M_0.

($i = 1$): If $A \in M_1$, then A is the consequent of a rule having no positive premise and thus $A \in \text{Potdef}(P)$.

Induction step: Assume that for each $i \leqslant k$ and for each proposition B, if $B \in M_i$ then $B \in \text{Potdef}(P)$. Consider $A \in M_{k+1}$ such that $\exists r \in P$, $A = \text{Cons}(r)$ and $\forall L \in \text{Prem}(r)$, $L \in M_k$. By induction hypothesis, $\forall L \in \text{Prem}(r)^+$, $L \in \text{Potdef}(P)$. Thus $A \in \text{Potdef}(P)$.

(ii) Let A be defined in P then $A \in E_P^k(\emptyset)$ for some k and by induction on k, we have that $A \in \bigcup_{i=0,\ldots,\infty} M_i$.

Thus $\text{Def}(P) \subseteq \bigcup_{i=0,\ldots,\infty} M_i$. □

Intuitively, if a proposition A is defined (respectively undefined) in the logic program P then one can reasonably expect the proposition A to be true (respectively, to be false) in the default model of P, if such a model exists. This property results from the preceding lemma.

Theorem 4.5. *Let P be a program over* Prop, *let A be in* Prop *and let M be a default model of P. Then*:

(i) *if $A \in M$ then A is potentially defined in P,*

(ii) *if A is defined in P then $A \in M$,*

(iii) *if A is undefined in P then $\neg A \in M$.*

Remark 4.6. It is interesting here to compare the notion of potentially defined proposition and the notion of supported proposition. Recall that a model M of P is a supported model iff for each true proposition A in M there exists a rule r in P with consequent A such that each premise of r is satisfied by M. Clearly, the notion of potentially defined proposition is weaker than the notion of supported proposition in the following sense: if A is true in M, where M is a supported model of P, then A is potentially defined in P. Besides, it should be recalled that a default model M of a logic program P is a supported model of P. Thus property (i) of Theorem 4.5 is immediate.

Proof. (i) and (ii) follow immediately from Lemma 4.4 and from the definition of a default model.

(iii) In order to see that $\neg A \in M$ if A is undefined, where M is a default model of P, we use property (i) and the fact that M is an interpretation, that is, a maximally consistent set of literals. □

For propositions that are potentially defined but not defined in a logic program P, nothing can be said about their "truth value" before one knows the default model of P. For instance, the program P of Example 4.2 has a default model $M = \{A, \neg B, C, \neg D, \neg E, \neg F, G\}$; the proposition C is potentially defined in P and "true" in M and the proposition D is potentially defined in P and "false" in M.

We now define two basic operations that transform a logic program into another one. The *Reduction* of P with respect to a literal L removes the premise L from each rule in P. The *Simplification* of P with respect to a literal L deletes each rule in P having L among its premises.

Definition 4.7. Let P be a logic program over Prop and L be a literal. Then:

$$\text{Reduce}(P, L) = \{r \mid s \in P, \text{Prem}(r) = \text{Prem}(s) - \{L\} \text{ and } \text{Cons}(r) = \text{Cons}(s)\},$$

$$\text{Simplify}(P, L) = P - \{r \mid L \in \text{Prem}(r)\}.$$

These two transformations correspond to Davis-Putman transformations [14] defined for sets of clauses.

Example 4.2 (*continued*)

$$P_1 = \text{Reduce}(P, A) = \{A, G, \neg A \to D, B \wedge \neg C \to D, \neg B \wedge \neg D \to C, E \to F, F \to E\},$$

$$P_2 = \text{Simplify}(P_1, \neg A) = \{A, G, B \wedge \neg C \to D, \neg B \wedge \neg D \to C, E \to F, F \to E\},$$

$$P_3 = \text{Reduce}(P_2, G) = \text{Simplify}(P_2, G),$$

$$P_4 = \text{Reduce}(P_3, \neg B) = \{A, G, B \wedge \neg C \to D, \neg D \to C, E \to F, F \to E\},$$

$$P_5 = \text{Simplify}(P_4, B) = \{A, G, \neg D \to C, E \to F, F \to E\},$$

$$P_6 = \text{Simplify}(P_5, E) = \{A, G, \neg D \to C, F \to E\},$$

$$P_7 = \text{Simplify}(P_6, F) = \{A, G, \neg D \to C\}.$$

Before formally stating the properties satisfied by Reduce and Simplify, we extend these operations in order to deal with sets of literals rather than with a single literal.

Definition 4.8. Let P be a logic program and S be a consistent set of literals. Then, the *Reduction of* P with respect to the set S of literals is recursively defined by:

$$\text{Reduce}(P, \emptyset) = P,$$

$$\text{Reduce}(P, S) = \text{Reduce}(\text{Reduce}(P, S - \{L\}), L) \text{ for some } L \text{ in } S.$$

In the same manner, the *Simplification of P* with respect to the set S of literals is recursively defined by:

$$\text{Simplify}(P, \emptyset) = P,$$

$$\text{Simplify}(P, S) = \text{Simplify}(\text{Simplify}(P, S - \{L\}), L) \text{ for some } L \text{ in } S.$$

The nature of each transformation, Reduction and Simplification, implies that their application to sets of literals is totally independent of the order of the elementary transformations performed with a single literal.

We now define the *effective* operator that transforms a logic program P by Reduction and Simplification with respect to the sets of defined and undefined propositions in P.

Intuitively, if a proposition A is defined in P then:

- on the one hand, an occurrence of A in the premise of a rule r is "solved" by the fact that "A is true for P" and thus the premise A may well be removed from the rule r without changing the meaning of the program P,
- on the other hand, an occurrence of $\neg A$ in the premise of a rule r is "unsolvable" because "A is true for P" and thus the rule r may well be removed from the program P without changing its meaning.

In the same manner, if the proposition A is undefined in P then (recall A is "false") removing $\neg A$ from the premises of each rule in P produces a program equivalent to P and deleting the rules in P having A among their premises also produces a program equivalent to P.

Definition 4.9. Let P be a logic program over Prop. The *effective operator* denoted by EFF is defined by:

$$\text{EFF}(P) = \text{Simplify}(\text{Reduce}(P, \text{Def}(P) \cup \neg.\text{Undef}(P)), \neg.\text{Def}(P) \cup \text{Undef}(P)).$$

Although in general Simplify and Reduce do not commute (consider the logic program $P = \{\neg B \rightarrow A\}$, then $\text{Simplify}(\text{Reduce}(P, \neg B), \neg B) = \{A\}$ and $\text{Reduce}(\text{Simplify}(P, \neg B), \neg B) = \emptyset$), note that in the definition of EFF the order of application of the two transformations is irrelevant because Simplify and Reduce have disjoint arguments concerning sets of literals.

The effective transformation $\text{EFF}(P)$ of P preserves the defined and the undefined propositions in P.

Property 4.10. *Let P be a logic program. Then:*

(i) *if the proposition A is defined in P then the proposition A is defined in* $\text{EFF}(P)$, *that is* $\text{Def}(P) \subseteq \text{Def}(\text{EFF}(P))$, *and*

(ii) *if the proposition A is undefined in P then the proposition A is undefined in* $\text{EFF}(P)$, *that is* $\text{Undef}(P) \subseteq \text{Undef}(\text{EFF}(P))$.

Proof. (i) It can be easily shown by induction on n that $E_P^n(\emptyset) \subseteq \mathrm{EFF}(P)$ for $n \geqslant 1$.

(ii) In order to prove that $\mathrm{Undef}(P) \subseteq \mathrm{Undef}(\mathrm{EFF}(P))$ we prove that $\mathrm{Potdef}(\mathrm{EFF}(P)) \subseteq \mathrm{Potdef}(P)$. Let us first note that if $A = \mathrm{Cons}(r)$ with $r \in \mathrm{EFF}(P)$ then $\exists r' \in P$ such that $\mathrm{Cons}(r') = \mathrm{Cons}(r)$ and $(\mathrm{Prem}(r')^+ - \mathrm{Prem}(r)^+) \subseteq \mathrm{Def}(P) \subseteq \mathrm{Potdef}(P)$. Hence if we know that $\mathrm{Prem}(r)^+ \subseteq \mathrm{Potdef}(P)$, then we can deduce that $\mathrm{Cons}(r') = \mathrm{Cons}(r) \in \mathrm{Potdef}(P)$. Using this fact it is straightforward to show by induction on n that $\mathrm{Potd}^n_{\mathrm{EFF}(P)} \subseteq \mathrm{Potdef}(P)$, where $(\mathrm{Potd}^n_{\mathrm{EFF}(P)})_{n \geqslant 1}$ is the sequence of potentially defined propositions in $\mathrm{EFF}(P)$ associated with $\mathrm{EFF}(P)$. □

Moreover, the effective transformation EFF preserves the semantics of programs.

Theorem 4.11. *Let P be a logic program. Then, P and* $\mathrm{EFF}(P)$ *are two equivalent logic programs, that is M is a default model of P iff M is a default model of* $\mathrm{EFF}(P)$.

Proof. In order to show that the programs P and $\mathrm{EFF}(P)$ are equivalent, it suffices to show that, given an interpretation M, M is a fixpoint of E_P iff M is a fixpoint of $E_{\mathrm{EFF}(P)}$. As a matter of fact, since the order of elementary transformations Reduce and Simplify is irrelevant, we show that:

(1) An interpretation M is a fixpoint of E_P iff M is a fixpoint of $E_{P'}$ where $P' = \mathrm{Reduce}(P, L_0)$ and L_0 is either a defined proposition in P or the negation of an undefined proposition in P.

(2) An interpretation M is a fixpoint of E_P iff M is a fixpoint of $E_{P'}$ where $P' = \mathrm{Simplify}(P, L_0)$ and L_0 is either the negation of a defined proposition in P or an undefined proposition in P.

Let us prove (1). Assume first that M is a fixpoint of E_P. By definition of $E_{P'}$, $M \subseteq E_{P'}(M)$. Now let us consider $A = \mathrm{Cons}(r')$ such that r' is in $P' = \mathrm{Reduce}(P, L_0)$ and $\forall L \in \mathrm{Prem}(r')$, $L \in M$. Then, either r' is in P or there exists a rule r in P such that $A = \mathrm{Cons}(r)$ and $\mathrm{Prem}(r) = \mathrm{Prem}(r') \cup \{L_0\}$. Now, by Theorem 4.5, because L_0 is either a defined proposition in P or the negation of an undefined proposition in P, $L_0 \in M$ thus $\forall L \in \mathrm{Prem}(r)$, $L \in M$ and $\mathrm{Cons}(r) = \mathrm{Cons}(r') = A$ is in $E_P(M) = M$. This implies that $E_{P'}(M) \subseteq M$. In conclusion, M is a fixpoint of $E_{P'}$.

Assume now that M is a fixpoint of $E_{P'}$. By definition of E_P, $M \subseteq E_P(M)$. Now let us consider $A = \mathrm{Cons}(r)$ such that r is in P and $\forall L \in \mathrm{Prem}(r)$, $L \in M$. Two cases arise. If r is in $P' = \mathrm{Reduce}(P, L_0)$ then immediately A is in M. Now if r is not in $P' = \mathrm{Reduce}(P, L_0)$, then there exists a rule r' in P' such that $\mathrm{Cons}(r) = \mathrm{Cons}(r')$ and $\mathrm{Prem}(r) = \mathrm{Prem}(r') \cup \{L_0\}$. Thus $\forall L \in \mathrm{Prem}(r)$, $L \in M$ implies that $\forall L \in \mathrm{Prem}(r')$, $L \in M$ and $\mathrm{Cons}(r) = \mathrm{Cons}(r') = A$ is in $E_{P'}(M) = M$. Thus $E_P(M) \subseteq M$. In conclusion, M is a fixpoint of E_P.

Let us now prove (2). Assume first that M is a fixpoint of $E_P(M)$. By definition of $E_{P'}$, $M \subseteq E_{P'}(M)$. Now let us consider $A = \mathrm{Cons}(r')$ such that r' is in $P' = \mathrm{Simplify}(P, L_0)$ and $\forall L \in \mathrm{Prem}(r')$, $L \in M$. Then, r' is in P and $A = \mathrm{Cons}(r')$ is in $E_P(M) = M$. Thus $E_{P'}(M) \subseteq M$. In conclusion, M is a fixpoint of $E_{P'}$.

Assume now that M is a fixpoint of $E_{P'}$. By definition of E_P, $M \subseteq E_P(M)$. Let us consider $A = \text{Cons}(r)$ such that r is in P and $\forall L \in \text{Prem}(r)$, $L \in M$. Assuming that r is not in $P' = \text{Simplify}(P, L_0)$ implies that, L_0 is a premise of r and either $\neg L_0$ is defined in P or L_0 is undefined in P. Thus by Theorem 4.5, the negation of L_0 is in M which leads to a contradiction with $\forall L \in \text{Prem}(r)$, $L \in M$. Thus r is in P' and $A = \text{Cons}(r)$ is in $E_{P'}(M) = M$. Thus $E_P(M) \subseteq M$. In conclusion, M is a fixpoint of E_P. □

At this point of the discussion, we are able, given a logic program P, to construct an equivalent program EFF(P) by utilizing the set Def(P) of defined propositions in P and the set Undef(P) of undefined in P. The new program obtained is syntactically simpler (and semantically equivalent to P). It contains less rules or rules with less premises. As suggested in Section 2, the idea is to construct from P a program equivalent to P and to get rid of as many as possible ineffective rules in P. If we examine the program P of Example 4.2, the program P_7 obtained by one application of the effective operator EFF on P, does not contain ineffective rules any more. As a matter of fact, note that P_7 is a stratifiable program. However, in the general case, ineffective rules may remain after one application of the effective operator EFF. An example of such a program is given below.

Example 4.12. Let us consider the logic program P defined as follows:

$$P = \{A0, \neg B0 \to A1, \neg A0 \to A0, \neg A1 \to A1, \neg A1 \to B1\}.$$

In the program P, $A0$ is defined, $A1$, $B1$ are potentially defined and $B0$ is undefined. Thus, $\text{EFF}(P) = \{A0, A1, \neg A1 \to A1, \neg A1 \to B1\}$. In EFF($P$), the two last rules are ineffective rules. In fact $A1$ is defined in EFF(P) and $\text{EFF}(\text{EFF}(P)) = \{A0, A1\}$ does not contain any more ineffective rules. In this case, $\text{EFF}^2(P)$ is even a very simple program containing only facts.

This leads us to consider successive applications of the effective operator EFF.

Definition 4.13. Let r and s be two rules. Let P and P' be two logic programs. Then:

(i) $r \leqslant s$ iff $\text{Cons}(r) = \text{Cons}(s)$ and $\text{Prem}(r) \subseteq \text{Prem}(s)$. We say that *r subsumes s.*

(ii) $P \leqslant P'$ iff $\forall r \in P - P'$ $\exists s \in P' - P$, $r \leqslant s$.

(iii) The sequence $\text{EFF}^i(P)$ is defined by $\text{EFF}^0(P) = P$ and for $i \geqslant 0$, $\text{EFF}^{i+1}(P) = \text{EFF}(\text{EFF}^i(P))$.

Intuitively at step $i+1$, $\text{EFF}^{i+1}(P)$ is obtained by Reduction and Simplification of $\text{EFF}^i(P)$ using the defined and undefined propositions in $\text{EFF}^i(P)$.

Property 4.14. *Let P be a logic program. Then:*

(i) *the relation* $\leqslant$ *defined among logic programs over* Prop *is a partial order,*

(ii) $\text{EFF}(P) \leqslant P$,

(iii) *there exists i such that* $\text{EFF}^i(P) = \text{EFF}^{i+1}(P)$.

Proof. (i) and (ii) are obtained by using the fact that we consider finite programs. In order to show (iii) let us define the size of a program P as the number of symbols occurring in P and let us denote it by $|P|$. Clearly if $P_{n+1} \preccurlyeq P_n$ and $P_n \neq P_{n+1}$ then $0 \leq |P_{n+1}| < |P_n|$. $\preccurlyeq$ is a well-founded ordering among finite programs. From (ii), it follows that if $\mathrm{EFF}^i(P) \neq \mathrm{EFF}^{i+1}(P)$ then $0 \leq |\mathrm{EFF}^{i+1}(P)| < |\mathrm{EFF}^i(P)|$. Hence there is some j such that $\mathrm{EFF}^j(P) = \mathrm{EFF}^{j+1}(P)$. □

In the following, we denote by $\mathrm{EFF}^\infty(P)$ the logic program $\mathrm{EFF}^i(P)$ such that $\mathrm{EFF}^i(P) = \mathrm{EFF}^{i+1}(P)$.

We are now ready to define a new class of "well-behaved" logic programs.

Definition 4.15. Let P be a logic program. Then, P is *effectively stratifiable* iff there exists i such that $\mathrm{EFF}^i(P)$ is stratifiable.

Theorem 4.16. *Let P be a logic program. P is effectively stratifiable entails that P has a unique default model M and $M = \mathrm{EFF}^\infty(P) \cup \neg.(\mathrm{Prop} - \mathrm{EFF}^\infty(P))$.*

Proof. The proof is immediate. From [3, 4], we know: $\mathrm{EFF}^i(P)$ stratifiable implies that $\mathrm{EFF}^i(P)$ has a unique default model and from Theorem 4.11, we have that P and $\mathrm{EFF}^i(P)$ are equivalent. Thus P has a unique default model. Now clearly, if there exists i such that $\mathrm{EFF}^i(P)$ is stratifiable then $\mathrm{EFF}^\infty(P)$ is a set of positive literals whose unique default model is $\mathrm{Def}(\mathrm{EFF}^\infty(P)) \cup \neg.\mathrm{Undef}(\mathrm{EFF}^\infty(P)) = \mathrm{EFF}^\infty(P) \cup \neg.(\mathrm{Prop} - \mathrm{EFF}^\infty(P))$. □

Example 4.2 (*continued*). The logic program P of Example 4.2 is effectively stratifiable because $\mathrm{EFF}(P) = P_7$ is stratifiable. Note that $\mathrm{EFF}^\infty(P) = \mathrm{EFF}^2(P) = \{A, C, G\}$ and that the unique default model of P is $M = \{A, \neg B, C, \neg D, \neg E, \neg F, G\}$.

We have exhibited a class of consistent and unambiguous logic programs that strictly includes the class of (locally) stratified programs with negation. Before showing with an example that the class of effectively stratifiable programs is a strict subclass of the class of logic programs that have a unique default model, we show that the class of effectively stratifiable programs cannot be augmented by deletion of subsumed rules.

Given a logic program P in which r subsumes s, deleting the rule s from P in order to simplify P is a very elementary and attractive idea. In fact:

Proposition 4.17. *Let P be a logic program. If r subsumes s in P then P and $P - \{s\}$ are equivalent logic programs (with respect to the default semantics).*

The proof is obvious and omitted here.

First of all, it is clear that the effective operator EFF does not eliminate subsumed rules (see Example 4.18). However, it is important to note here that combining

deletion of subsumed rules with the two other elementary transformations of Reduction and Simplification has no effect on the definition of effectively stratifiable logic programs.

Example 4.18. Consider the following logic program.

$$P = \{\neg A \wedge \neg B \to C, \neg B \to C, \neg C \to A, \neg C \to B\}.$$

The program P is irreducible: A, B and C are all potentially defined propositions in P and none of them is defined in P. Thus $\mathrm{EFF}(P) = P$. The first rule $\neg A \wedge \neg B \to C$ in P is subsumed by the second one $\neg B \to C$. Note that the logic program P is not (effectively) stratifiable and that the logic program $P' = \{\neg B \to C, \neg C \to A, \neg C \to B\}$ obtained by removing the subsumed rule $\neg A \wedge \neg B \to C$ from P is irreducible and not (effectively) stratifiable.

Deletion of subsumed rules does not augment the class of effectively stratifiable logic programs. In order to formally state this, we introduce an operator Sub on logic programs to eliminate subsumed rules.

Definition 4.19. Let P be a logic program. $\mathrm{Sub}(P)$ is the subset of P such that:

(i) $\forall r, r' \in \mathrm{Sub}(P)$ r does not subsume r',

(ii) $\forall r \in P\ \exists r' \in \mathrm{Sub}(P)$ r is subsumed by r' (i.e. $r' \leqslant r$).

Note that $\mathrm{sub}(P) \leqslant P$.

Now, we consider the sequence $(\text{sub-eff}^i(P))_{i \geqslant 0}$ defined by

$$\text{sub-eff}^0(P) = \mathrm{Sub}(P),$$

$$\text{sub-eff}^{i+1}(P) = \mathrm{Sub}(\mathrm{EFF}(\text{sub-eff}^i(P))), \quad \text{for } i \geqslant 0.$$

Recall that $\mathrm{EFF}(P) \leqslant P$. Therefore $(\text{sub-eff}^i(P))_{i \geqslant 0}$ is a decreasing sequence of finite programs and thus there exists j such that $\text{sub-eff}^j(P) = \text{sub-eff}^{j+1}(P)$. Let us denote by $\text{sub-eff}^\infty(P)$ the first element $\text{sub-eff}^j(P)$ of the sequence $(\text{sub-eff}^i(P))_{i \geqslant 0}$ such that $\text{sub-eff}^j(P) = \text{sub-eff}^{j+1}(P)$.

Theorem 4.20. *Let P be a logic program. If* $\text{sub-eff}^\infty(P)$ *is a stratifiable program then P is effectively stratifiable* (*i.e.* $\mathrm{EFF}^\infty(P)$ *is stratifiable*).

Proof. First we show that $\forall j\ \exists i\ \mathrm{Sub}(\mathrm{EFF}^i(P)) \leqslant \text{sub-eff}^j(P)$. This can be easily established by induction on j by using Property 4.14 and the two straightforward properties:

(1) $\mathrm{Def}(P) = \mathrm{Def}(\mathrm{Sub}(P))$ and $\mathrm{Undef}(P) = \mathrm{Undef}(\mathrm{Sub}(P))$,

(2) $\mathrm{Sub}(\mathrm{EFF}(P)) \leqslant \mathrm{EFF}(\mathrm{Sub}(P))$ (note that the converse is not true).

It follows from this that $\mathrm{Sub}(\mathrm{EFF}^\infty(P)) \leqslant \text{sub-eff}^\infty(P)$.

Now clearly, if P_1 is stratifiable and if $P_2 \leqslant P_1$ then P_2 is stratifiable. Thus if $\text{sub-eff}^\infty(P)$ is stratifiable then $\mathrm{Sub}(\mathrm{EFF}^\infty(P))$ is stratifiable. It remains to show that

$\text{EFF}^\infty(P)$ is stratifiable. $\text{Def}(\text{EFF}^\infty(P)) = \text{Def}(\text{Sub}(\text{EFF}^\infty(P)))$, $\text{Undef}(\text{EFF}^\infty(P)) = \text{Undef}(\text{Sub}(\text{EFF}^\infty(P)))$ and $\text{EFF}^\infty(P)$ irreducible entail that $\text{Sub}(\text{EFF}^\infty(P))$ is irreducible. Now because $\text{Sub}(\text{EFF}^\infty(P))$ is irreducible and stratifiable, each proposition is either defined in $\text{Sub}(\text{EFF}^\infty(P))$ (and thus in $\text{EFF}^\infty(P)$) or undefined in $\text{Sub}(\text{EFF}^\infty(P))$ (and thus in $\text{EFF}^\infty(P)$). Consequently $\text{EFF}^\infty(P)$ is stratifiable. □

We now show that the constraint of effective stratification is a sufficient but not a necessary condition for the uniqueness of default model of logic programs (with respect to the default logic semantics). The next example presents a program P which is not effectively stratifiable but has a unique default model.

Example 4.21. Let us consider the following logic program:

$$P = \{\neg A \to B, \neg C \to A, \neg B \wedge \neg A \to C\} \ (\text{Prop} = \{A, B, C\}).$$

The program P is irreducible, i.e. $\text{EFF}^\infty(P) = P$ because $\text{Def}(P)$ and $\text{Undef}(P)$ are both empty. $P = \text{EFF}^\infty(P)$ is obviously not stratifiable. However, the logic program P above has a unique default model, namely $\{A, \neg B, \neg C\}$.

Let us show quickly that the default model associated with P captures the intended meaning of P. In fact here, because the first rule $\neg A \to B$ is the only rule having B in its consequence, the negative premise $\neg A$ of the third rule can be "replaced" by B. The third rule becomes $\neg B \wedge B \to C$. This latter rule is irrelevant and can thus be removed. We end up with a program of the form $\{\neg A \to B, \neg C \to A\}$ that is stratifiable and has a unique default model namely $\{A, \neg B, \neg C\}$.

We briefly suggest how the previous definition of effectively stratifiable programs could be extended in the case where the logic program P is an infinite set of rules defined over an infinite set Prop of propositions. Intuitively, an infinite propositional program corresponds to the instantiation of a first-order logic program with function symbols.

Thus we could consider Definition 4.15 to be extended in the following straightforward way: P is *effectively stratifiable* iff there exists an ordinal α such that $\text{EFF}^\alpha(P)$ is stratifiable, where $\text{EFF}^\alpha(P) = \text{greatest\_lower\_bound}(\{\text{EFF}^\beta(P)/\beta < \alpha\})$.

The problem encountered here is that, given a logic program P, it may happen that the ordinal α such that $\text{EFF}^\alpha(P)$ is stratifiable is above ω. Such a situation is of no interest for practical reasons. As written in [35], "iterating even to ω takes too long". In [6] we propose two examples of first-order logic programs with function symbols: the closure ordinal of the first program is ω while the closure ordinal for the second one is $\omega + 1$.

Proposition 4.22. *Let P be a first-order logic program with functions. Then, P is locally stratifiable entails that P is effectively stratifiable.*

Indeed, if P is locally stratifiable then the instantiation of P is stratifiable, and thus effectively stratifiable. That locally stratifiable programs have a unique default model is shown in [3, 4].

An algorithm for testing effective stratifiability for a logic program P is provided in [6]. It is a forward procedure that generates Def(P) and Potdef(P), and propagates defined and undefined propositions in a breadth-first fashion. (It is analogous to the linear algorithm proposed by [13] for testing the satisfiability of propositional Horn formulae.)

It is worth noting that this algorithm computes the default model of an effectively stratifiable program in polynomial time, while determining in general whether a logic program has a default model is a NP-complete problem [6].

5. Comparison with other approaches

In this section, we briefly compare the class of effectively stratifiable logic programs and the class of weakly stratifiable logic programs [29]. We also discuss weakly perfect model semantics [29], unique stable semantics [19] and well-founded semantics [35] versus the default logic semantics for logic programs with negation.

5.1. Weak stratification and effective stratification

As pointed out in Section 2, the perfect model semantics is unable to capture the intended meaning of some simple and very useful logic programs which are not (locally) stratifiable but still have a clear intended meaning. In [29], a new semantics, called weakly perfect model semantics, is introduced as a natural extension of perfect model semantics and the class of weakly stratifiable programs is proposed as enlarging the class of locally stratifiable programs. As for perfect models and stratification, the definition of weakly perfect models is strongly linked to the definition of weak stratification.

In order to properly carry on our comparative study, we need to briefly recall here the definitions of weakly perfect models and weakly stratifiable programs [29].

In the following, P is a logic program defined over the set Prop of propositions. We define the *precedence graph* G_P associated with P as follows. There exists a positive (respectively negative) edge from B to A in G_P iff there exists a rule r in P with $\text{Cons}(r) = A$ and B is in $\text{Prem}(r)^+$ (respectively B is in $\text{Prem}(r)^-$).

Definition 5.1 (Przymunsinska and Przymunsinski [29]). (1) The *dependency* (*or priority*) *relations* $<$ and $\leqslant$ between elements of Prop are defined by

(i) $A \leqslant B$ iff there is a directed path from A to B, and

(ii) $A < B$ iff there is a directed path from A to B passing through a negative edge.

(2) The *equivalence relation* $\approx$ between elements of Prop is defined by

$$A \approx B \text{ iff } A = B \text{ or } (A < B \text{ and } B < A).$$

Equivalence classes are called *components* of the graph G_P and a component is trivial if it consists of one point.

(3) A relation $<$ is introduced between components of the precedence graph G_P:

$$C1 < C2 \text{ iff } C_1 \neq C_2 \quad \text{and} \quad \exists A \in C_1 \; \exists B \in C_2 \text{ such that } A < B.$$

This relation induces a *partial order.* A component C_1 is *minimal* if there is no component C_2 such that $C_2 < C_1$.

(4) The *bottom stratum* $S(P)$ of P is the union of the minimal components of G_P.

(5) The *bottom layer* $L(P)$ of P is the subprogram consisting of rules whose consequents belong to the bottom stratum $S(P)$ of P.

Intuitively, given a logic program P, its weakly perfect model is defined as follows. First, the bottom layer $L(P)$ of P is considered and the least model of $L(P)$ is computed. Then, this least model of $L(P)$ serves to prune the initial program. The process is iterated by considering the bottom layer of the new simplified logic program, ...

In order to make the formal definition clear, let us recall some technical definitions. Let P be a logic program defined over Prop, a model M of P (that is, a maximally consistent set of literals over Prop satisfying P) is the least model of P iff the positive part of M is equal to the intersection of the positive parts of all models of P, that is, $M^+ = \bigcap_{M' \text{model of } P} M'^+$.

Recall that if P is a positive logic program (a program without negation), then P has a least model. Otherwise, P may not have a least model.

In the following definition it is assumed that in P, there is no rule that is subsumed by a unit rule in P (a unit rule is a proposition). If it is not the case, these subsumed rules are removed from P.

Definition 5.2 (Przymunsinska and Przymunsinski [29]). (1) Let $P_0 = P$, Prop_0 be the set of propositions having no occurrence in P, $L_0 = \emptyset$ and $N_0 = \neg.\text{Prop}_0$. Then, for $k > 0$, let P_k, L_k, Prop_k, N_k be defined by the following: P_k is obtained by removing from $\text{Simplify}(\text{Reduce}(P_{k-1}, N_{k-1}), \neg.N_{k-1})$ all the rules whose consequents are in N_{k-1}, $L_k = L(P_k)$, Prop_k is the set of propositions occurring in L_k and,

if $P_k = \emptyset$, then $N_k = N_{k-1}$ else

if the program L_k over Prop_k has no least model then $N_k = N_{k-1}$

else N_k is the least model of the program L_k over Prop_k.

Now let k be such that $N_k = N_{k+1}$ and let $N_P = \bigcup_{i=0,\ldots,k} N_i$.

if $P_k = \emptyset$, then N_P is the *weakly perfect model* of P, and

if $P_k \neq \emptyset$, then N_P is the *partial weakly perfect model* of P.

(2) P is *weakly stratifiable* iff

(a) all of its strata S_i $(i \geq 1)$ consist only of trivial components or, equivalently,

(b) all of its layers L_i $(i \geq 1)$ are positive logic programs.

Remark 5.3. The above definition of weak stratification is ambiguous simply because the statements (a) "all of its strata S_i consist only of trivial components" and (b) "all of its layers L_i are positive logic programs" are not always equivalent. To see that (a) and (b) are not equivalent, it suffices to consider the logic program $P_0 = \{\neg A \rightarrow A\}$.

Following the alternative (a) of the definition, the program is weakly stratifiable. Its stratum $S_1 = \{A\}$ is a trivial component. Following the alternative (b) of the definition, the program is not weakly stratifiable because the layer $L_1 = P_0 = \{\neg A \rightarrow A\}$ is not a positive program.

In the following, we assume that weak stratification is defined by (b) and compare the class of effectively stratifiable programs and the class of weakly stratifiable programs.

Proposition 5.4. *Let P be a propositional logic program. Then, if P is weakly stratifiable and N_P is the weakly perfect model of P then P is effectively stratifiable and N_P is the unique default model of P.*

Sketch of proof. In order to prove Proposition 5.4, it suffices to show that each proposition true (resp. false) in N_P is a defined (resp. undefined) proposition in $\mathrm{EFF}^{\infty}(P)$. Assuming this is the case, then:

- on the one hand, because N_P is maximally consistent, $\mathrm{EFF}^{\infty}(P) = N_P^+$ is stratifiable, thus P is effectively stratifiable, and
- on the other hand, because P is effectively stratifiable, by Theorem 4.16, the default model of P is equal to $\mathrm{Def}(\mathrm{EFF}^{\infty}(P)) \cup \neg.\mathrm{Undef}(\mathrm{EFF}^{\infty}(P)) = \mathrm{EFF}^{\infty}(P) \cup \neg.(\mathrm{Prop} - \mathrm{EFF}^{\infty}(P))$ and thus N_P is the default model of P.

Showing that each proposition true (resp. false) in $N_P = \bigcup_{i=0,\ldots,k} N_i$ is a defined (resp. undefined) proposition in $\mathrm{EFF}^{\infty}(P)$ is done by induction on i, for each N_i. The proof is easy and omitted here. □

This proposition follows also from other results presented at the end of this section.

Example 5.6 presents a logic program that is effectively stratifiable but not weakly stratifiable. This example shows that:

Theorem 5.5. *The class of effectively stratifiable logic programs is strictly larger than the class of weakly stratifiable logic programs.*

Example 5.6. Consider the program of Example 4.3

$$P = \{A \rightarrow B,\ B \rightarrow A,\ \neg A \rightarrow C,\ \neg C \wedge B \rightarrow A\}.$$

(1) $\mathrm{Def}(P) = \emptyset$; $\mathrm{Potdef}(P) = \{C\}$; $\mathrm{Undef}(P) = \{A, B\}$. Thus $\mathrm{EFF}(P) = \{C\}$ is stratifiable and the program P is effectively stratifiable. Its default model is $\{\neg A, \neg B, C\}$.

(2) P has a unique minimal component $\{A, B, C\}$. The bottom layer $L(P)$ of P is P itself and is not a positive logic program. Thus P is not weakly stratifiable. Since the bottom layer $L(P) = P$ of P has no least model the partial weakly perfect model of P is empty and P has no weakly perfect model.

The gap between the class of effectively stratifiable programs and the class of weakly stratifiable programs is explained by the gap between the set of defined or undefined propositions in a program P and the contents of the minimal components of the graph G_P associated with the program P. For instance, in the case of Example 5.6, the analysis of the precedence graph G_P does not allow to isolate the propositions A and B from the proposition C (all three propositions are in the connected component of the graph). The notion of potentially defined propositions allows us to exhibit that although the definition of A depends on the negation of C and the definition of C depends on the negation of A, there is no chance that the rule $\neg C \wedge B \rightarrow A$ could be activated. This is due to the fact that the rule $\neg C \wedge B \rightarrow A$ defining A using $\neg C$ uses B positively and to the fact that A and B are mutually positively dependent. This leads us to say that A and B are undefined in P.

5.2. Weakly perfect model semantics and default model semantics

While some effectively stratifiable programs have no weakly perfect model, there are, as shown by the following example, some logic programs that are not weakly stratifiable, still have a weakly perfect model, but do not have a default model.

Example 5.7. Consider the following logic program P

$$P = \{\neg A \rightarrow B,\ B \rightarrow A\}.$$

(1) The propositions A and B are both potentially defined in P. $\mathrm{EFF}(P) = P$ and thus P is not effectively stratifiable. It is easy to see that P has no default model.

(2) P has a unique minimal component $\{A, B\}$.

The bottom layer $L(P)$ of P is P itself and is not a positive logic program. Thus P is not weakly stratifiable. However the bottom layer $L(P) = P$ of P has a least model $N_P = \{A, \neg B\}$ which is the weakly perfect model of P.

In fact in the preceding example, negation is interpreted as first-order negation. The weakly perfect model N_P is in fact the least model of $\{A \vee B, \neg B \vee A\}$. It seems to us that it can be confusing to deal with (that is difficult to understand and also to manage) a logic program containing two kinds of semantics for the same symbol of negation.

Note also that, although N_P is a minimal model of P, it is not a supported model of P. Roughly speaking this means that N_P cannot reproduce itself from the rules in P.

Now we compare the default model semantics with the stable model semantics and the well-founded semantics.

5.3. Stable model semantics and default model semantics

First we show that stable models introduced in [19] as a new semantics for logic programs are equivalent to default models. The default model semantics was previously introduced in [3, 4]. The equivalence of default models and stable models is not surprising. The definition of the former is based on default logic [30], while the definition of the latter is based on another nonmonotonic logic, namely autoepistemic logic [25]. Recently, Konolige [20] has shown the equivalence of default logic and autoepistemic logic.

We recall the definition of stable model.

Definition 5.8 (Van Gelder et al. [35]). Let P be a propositional logic program and let M be a model of P. Let M' be defined by the following three transformations:

(1) Define $P_1 = T_1(P, M)$ to be the transformation by which every rule with a negative premise that is inconsistent with M is discarded; the output of the transformation is the set of rules that remain.

(2) Define $T_2(P)$ to be the transformation by which all negative premises are dropped from rules of P, leaving a Horn program. We call $P_2 = T_2(T_1(P, M))$, the *reduction of P with respect to M.*

(3) Form M', the minimal model of $P_2 = T_2(T_1(P, M))$, in the sense of [33]. By "minimal", we mean the set of positive literals is minimized.

Call M a *stable model* if $M = M'$.

Theorem 5.9. *Let P be a logic program. M is a stable model of P iff M is a default model of P.*

Proof. Note that if a rule in P has a negative premise which is inconsistent with M, then this rule cannot be used in order to construct $E'_P(M^-)$. In other words, $E'_P(M^-) = E'_{P_1}(M^-)$.

If each negative premise of a rule of P_1 is consistent with M, i.e. if each negative premise of a rule of P_1 is in M^-, then clearly the consequent of this rule is in $E'_{P_1}(M^-)$ iff it is in $E'_{P_2}(M^-)$.

Since P_2 is a Horn program, $E'_{P_2}(M^-) = E^i_{P_2}(\emptyset) \cup M^-$. Let M' be the minimal model of P_2. The set of positive literals of M' is $\bigcup_{i=0,\ldots,\infty} E^i_{P_2}(\emptyset)$ [2]. Therefore the set of positive literals of M' is $\bigcup_{i=0,\ldots,\infty} E^i_{P_2}(M^-) - M^- = \bigcup_{i=0,\ldots,\infty} E^i_P(M^-) - M^-$. From this we deduce that $M' = M$ iff $M = \bigcup_{i=0,\ldots,\infty} E^i_P(M^-)$, that is M' is a stable model of P iff M' is a default model of P. □

In [35], stable models are compared with well-founded models. This comparison applies to default models versus well-founded models because of the equivalence between default and stable models. These results are recalled below. In the following

we enrich the comparison with a new result. Effective stratification characterizes the class of logic programs that have a 2-valued well-founded model.

5.4. Well-founded semantics and default model semantics

In order to present well-founded models, we recall the concept of unfounded sets.

Definition 5.10 (Van Gelder et al. [35]). Let S be a set of propositions and I be a consistent set of literals (also called a partial interpretation). S is an *unfounded set of P with respect to I* if each A in S satisfies the following. For each rule r in P such that $\text{Cons}(r) = A$, (at least) one of the following holds:

(1) there is an L in $\text{Prem}(r)$ such that L is in $\neg.I$, i.e. L is inconsistent with the partial interpretation I.

(2) there is a B in $\text{Prem}(r)^+$ such that B is in S.

As noted in [35], a literal that makes (1) or (2) true is a witness of unusability of the rule r (with respect to I).

We restrict the general definition of well-founded models given in [35] to the case where only finite propositional programs are considered. The well-founded model of P is defined as the limit of a sequence of partial interpretations. This sequence is constructed starting from the empty interpretation by iteratively adding to it consequents of rules whose premises are in the current partial interpretation and the negation of the propositions in the greatest unfounded set of P with respect to the current interpretation.

Definition 5.11 (Van Gelder et al. [35]). Let U_p be the operator defined over sets of literals by: $U_p(I) = \neg.G$, where G is the greatest unfounded set of propositions of P with respect to I. Let us consider the sequence of sets of literals $(I_k)_{k \geqslant 0}$ defined as follows: $I_0 = \emptyset$ and for $k \geqslant 0$, $I_{k+1} = E_p(I_k) \cup U_p(I_k)$ where E_p has been defined in Section 3. Note that each I_k is consistent and the monotonic sequence $(I_k)_{k \geqslant 0}$ reaches a limit I^* after a finite ordinal. If I^* is maximally consistent then I^* is a model of P called the *well-founded model of P*; otherwise I^* is called the *well-founded partial model of P*.

Result 5.12 (Van Gelder et al. [35]). Let P be a logic program.

(a) Any stable model M (and any default model M) of P contains the well-founded partial model I^* of P.

(b) If P has a well-founded model M then M is the unique stable model (and M is the unique default model) of P.

(c) If P is locally stratifiable, then it has a well-founded model which is identical to the perfect model of P (i.e. to the unique default model of P).

We will show that this last result can be generalized to effectively stratifiable logic programs (Theorem 5.15).

Just as we have defined Undef(P) as the complement of Potdef(P), we can introduce the set of potentially founded propositions in P with respect to I, denoted by Potfound(P, I) as the complement of the greatest unfounded set of P with respect to I.

Definition 5.13. Let Potfound(P, I) be the set of *potentially founded propositions in P with respect to I.* $\text{Potfound}(P, I) = \bigcup_{i=0,\ldots,\infty} \text{Potf}(P, I)^i$, where

$$\text{Potf}(P, I)^0 = \{\text{Cons}(r) \mid r \in P, \text{Prem}(r)^+ = \emptyset \text{ and } \forall L \in \text{Prem}(r)^-, L \notin \neg.I\}$$

$$\text{Potf}(P, I)^{i+1} = \text{Potf}(P, I)^i \cup \{\text{Cons}(r) \mid r \in P, \forall L \in \text{Prem}(r)\, L \notin \neg.I \text{ and } \forall B \in \text{Prem}(r)^+ B \in \text{Potf}(P, I)^i\}.$$

It is important here to note that the notion of potentially founded propositions is the dual of the notion of unfounded propositions.

Theorem 5.14. *Let P be a logic program. Then*

(1) $\text{Potdef}(P) = \text{Potfound}(P, \emptyset)$, *and*

(2) $U_P(I) = \neg.(\text{Prop} - \text{Potfound}(P, I))$.

Proof. The proof of (1) is immediate.

In order to prove (2), we proceed in two steps. First we show that $\neg.U_P(I)$ and Potfound(P, I) are disjoint and then we show that (Prop − Potfound(P, I)) is an unfounded set of P with respect to I. This immediately leads to the conclusion that $\neg.U_P(I)$, the greatest unfounded set of P with respect to I, is equal to Prop − Potfound(P, I).

In order to prove that $\neg.U_P(I)$ and Potfound(P, I) are disjoint, we show by induction on i that if $A \in \text{Potf}(P, I)^i$ then $A \notin \neg.U_P(I)$. The initial step ($i = 0$) is immediate. Assume that for $i \leqslant k$, $A \in \text{Potf}(P, I)^i$ implies $A \notin \neg.U_P(I)$ and let us consider $A \in \text{Potf}(P, I)^{k+1}$. Then there exists a rule r in P such that $\text{Cons}(r) = A$, $\forall L \in \text{Prem}(r)$ $L \notin \neg.I$ and $\forall B \in \text{Prem}(r)^+$ $B \in \text{Potf}(P, I)^k$. By induction hypothesis, we have $\forall B \in \text{Prem}(r)^+$ $B \notin \neg.U_P(I)$ and thus $A \notin \neg.U_P(I)$.

Now let us prove that Prop − Potfound(P, I) is an unfounded set of P with respect to I. Let $A \in \text{Prop} - \text{Potfound}(P, I)$, let r be a rule in P such that $\text{Cons}(r) = A$. Then because $A \notin \text{Potfound}(P, I)$, either there exists $L \in \text{Prem}(r)^+$ such that $L \notin \text{Potfound}(P, I)$, i.e. there exists $L \in \text{Prem}(r)^+$ such that $L \in \text{Prop} - \text{Potfound}(P, I)$ or there exists $L \in \text{Prem}(r)$ such that $L \in \neg.I$. Thus Prop − Potfound(P, I) is unfounded. □

It is important to note here that the sequence $(\text{Potf}(P, I)^i)_{i \geqslant 0}$ provides a constructive definition of the greatest unfounded set of P with respect to a partial interpretation I and thus a constructive definition of the well-founded model of P.

It is also interesting to note that effective stratification characterizes the class of logic programs that have a (total) well-founded model.

Theorem 5.15. *Let P be a logic program. P is effectively stratifiable iff the well-founded partial model of P is a well-founded model. Moreover, for effectively stratifiable programs, default model and well-founded model coincide.*

Proof. In order to prove this result, we are going to slightly modify the way well-founded models are defined. This modification provides a sequence of interpretations which is exactly the sequence of sets of defined plus undefined propositions in the programs $\mathrm{EFF}^i(P)$.

Let P be a logic program. Let $(I_i)_{(i\geqslant 0)}$ be the sequence defined in Definition 5.11. Recall that the limit of this sequence is denoted by I^*. Now let us consider the sequence $(J_i)_{(i\geqslant 0)}$ defined by: $J_0=\emptyset$, and $J_{i+1}=E_p\uparrow\omega(J_i)\cup U_p(J_i)$ for $i\geqslant 0$, where $E_p\uparrow\omega(J_i)=\bigcup_{k=1,\ldots,\infty} E_P^k(J_i)$. Note that, because E_p and U_p are monotonic, the sequence $(J_i)_{(i\geqslant 0)}$ is increasing. We denote by J^* the limit of this sequence.

Let us now show that these two sequences have the same limit, i.e. that $I^*=J^*$. Clearly, because E_p and U_p are monotonic, $I_i\subseteq J_i$ for each $i\geqslant 0$. (This is obviously true for $i=0$, and then, $I_i=E_p(I_{i-1})\cup U_p(I_{i-1})\subseteq E_p(J_{i-1})\cup U_p(J_{i-1})\subseteq E_p\uparrow\omega(J_{i-1})\cup U_p(J_{i-1})=J_i$.) Hence, $I^*\subseteq J^*$.

It remains to show that for each i, $J_i\subseteq I^*$. This is straightforward from the fact that I^* is the limit of the sequence $(I_i)_{(i\geqslant 0)}$ and thus $E_p\uparrow\omega(I^*)=I^*$ and $U_p(I^*)=I^*$. (It suffices to notice that $J_0\subseteq I^*$, and to proceed by induction.) Thus $I^*=J^*$.

Now in order to conclude, it suffices to establish the correspondence between the sequence $(J_i)_{(i\geqslant 1)}$ and the sequence $(K_i)_{(i\geqslant 1)}$ defined by

$$K_i=\mathrm{Def}(\mathrm{EFF}^{i-1}(P))\cup\neg.\mathrm{Undef}(\mathrm{EFF}^{i-1}(P)).$$

We naturally proceed by induction.

($i=1$): On the one hand, Theorem 5.14 entails that $\mathrm{Undef}(P)=\neg.U_p(\emptyset)$. On the other hand, by definition of $\mathrm{Def}(P)$, we have $\mathrm{Def}(P)=E_p\uparrow\omega(\emptyset)$. Thus, $K_1=J_1$. For the induction step it suffices to show that given a program P and $I=\mathrm{Def}(P)\cup\neg.\mathrm{Undef}(P)$, $\mathrm{Def}(\mathrm{EFF}(P))=E_p\uparrow\omega(I)$ and $\neg.\mathrm{Undef}(\mathrm{EFF}(P))=U_p(I)$. This is immediate from the definitions of EFF, Def, Potdef and U_p.

Thus the sequences $(J_i)_{(i\geqslant 1)}$ and $(K_i)_{(i\geqslant 1)}$ are the same. This immediately gives that P is effectively stratifiable iff the well-founded model of P is total (maximally consistent) and that, if P is effectively stratifiable, its default model is equal to its well-founded model. □

Remark 5.16. It is shown in [29] that every weakly stratifiable program has a well-founded model and that its weakly perfect model is also its well-founded model. From this and from Theorem 5.15 we again obtain Proposition 5.4.

However the well-founded semantics and the default semantics do not completely match. The example below due to [35] proposes a logic program that has an empty partial well-founded model but does have a default model.

Example 5.17. Consider the following logic program:

$$P = \{\neg B \to A, \neg A \to B, \neg C \to C, \neg B \to C\}.$$

(1) The set Def(P) of defined propositions in P and the set Undef(P) of undefined propositions in P are empty. EFF(P) = P is not stratifiable and thus P is not effectively stratifiable. However P has a unique default model $M = \{A, \neg B, C\}$. Roughly speaking, if M is a default model of P then C has to belong to M because of the third rule (because it is impossible to have $\neg C$). Now, we need to justify that C is in M. This can be done using the last rule and assuming B to be false. Finally, assuming that B is false is consistent with the program and the truth value already assigned to C. The first rule entails that A is in M. The second rule of P is not used.

(2) The greatest unfounded set of propositions of P with respect to the empty partial interpretation is empty. Nothing can be derived using P from the empty partial interpretation. Thus the partial well-founded model of P is the empty set.

Acknowledgment

Many thanks to Serge Abiteboul, Serenella Cerrito, Stephane Grumbach, Evangelos Paschos and Victor Vianu for useful suggestions and discussions. Work performed by N. Bidoit was partially supported by the PRC BD3 national project. Work performed by C. Froidevaux was partially supported by the PRC IA national project.

References

[1] R.K. Apt, H. Blair and A. Walker, Towards a theory of declarative knowledge, in: *Workshop on Foundations of Deductive Databases and Logic Programming* (1986) 546-628.

[2] K.R. Apt and M.H. Van Emden, Contributions to the theory of logic programming, *J. ACM* **29** (1982) 841-862.

[3] N. Bidoit and C. Froidevaux, Minimalism subsumes default logic and circumscription in stratified logic programming, in: *Proc. Logic In Computer Science* (IEEE, New York, 1987) 89-97.

[4] N. Bidoit and C. Froidevaux, Declarative semantics of stratified logic programs and databases: minimalism subsumes default logic and circumscription, *Inform. and Comput.*, to appear.

[5] N. Bidoit and C. Froidevaux, More on stratified default theories, in: *Proc. European Conf. on Artificial Intelligence*, Munich (1988) 492-494.

[6] N. Bidoit and C. Froidevaux, Negation by default and non stratifiable programs, Research Report 437, LRI, Orsay, 1988.

[7] N. Bidoit and R. Hull, Positivism vs minimalism in deductive databases, in: *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems* (1986) 123-132.

[8] N. Bidoit and R. Hull, Minimalism, justification and non-monotonicity in deductive databases, *J. Comput. System Sci.* **38** (2) (1989) 290-325.

[9] G. Bossu and P. Siegel, Saturation, nonmonotonic reasoning and the closed world assumption, *Artificial Intelligence* **25** (1985) 13-63.

[10] K.L. Clark, Negation as failure, in: H. Gallaire and J. Minker, eds., *Logic and Data Bases* (New York, 1978) 293-322.

[11] A.K. Chandra and D. Harel, Horn clause queries and generalizations, *J. Logic Programming* **2**(1) (1985) 1-15.
[12] C.-L. Chang and R.C.-T. Lee, *Symbolic Logic and Mechanical Theorem Proving, Computer Science and Applied Mathematics* (Academic Press, New York, 1973).
[13] W.F. Dowling and J. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, *J. Logic Programming* **3** (1984) 267-284.
[14] M. Davis and H. Putnam, A computing procedure for quantification theory, *J. ACM* **7**(3) (1960) 201-215.
[15] D. Etherington, A semantics for default logic, in: *Proc. IJCAI*, Milano (1987) 495-498.
[16] H.B. Enderton, *A Mathematical Introduction to Logic* (Academic Press, New York, 1972).
[17] R. Gallier, *Logic For Computer Science, Foundations of Automatic Theorem Proving* (Harper & Row, New York, 1986).
[18] M. Gelfond, On stratified autoepistemic theories, in: *Proc. AAAI-87* (1987) 207-211.
[19] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, in: *Proc. Logic Programming Conf.* Seattle (1988).
[20] K. Konolige, On the relation between default and autoepistemic logic, *Artificial Intelligence* **35**(3) (1988) 343-382.
[21] P.G. Kolaitis and C.H. Papadimitriou, Why not negation by fixpoint?, in: *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems* (1988) 231-239.
[22] J.W. Lloyd, *Foundations of Logic Programming*, 2nd edn. (Springer-Verlag, New York, 1987).
[23] V. Lifschitz, On the declarative semantics of logic programs with negation, in: *Workshop on Foundations of Deductive Databases and Logic Programming* (1986) 420-432.
[24] P. Mancarella, S. Martini and D. Pedreshi, Complete logic programs with domain-closure axiom, *J. Logic Programming* **5** (1988) 263-276.
[25] R.C. Moore, Semantic considerations on non-monotonic logic, *Artificial Intelligence* **25** (1985) 75-94.
[26] S. Naqvi, A logic for negation in database systems, in: *Workshop on Foundations of Deductive Databases and Logic Programming* (1986) 378-387.
[27] T.C. Przymunsinski, On the semantics of stratified deductive databases, in: *Workshop on Foundations of Deductive Databases and Logic Programming* (1986) 433-443.
[28] T.C. Przymunsinski, On the relationship between logic programming and non-monotonic reasoning, in: *Proc. AAAI-88* (1988) 444-448.
[29] H. Przymunsinska and T.C. Przymunsinski, Weakly perfect model semantics for logic programs, in: *Proc. Logic Programming Conf.*, Seattle (1988) 1106-1120.
[30] R. Reiter, A logic for default reasoning, *Artificial Intelligence* **13**(1) (1980) 81-132.
[31] J.C. Shepherdson, Negation as failure, II, *J. Logic Programming* **2**(3) (1985) 185-200.
[32] J.C. Shepherdson, Negation in logic programming, in: J. Minker, ed., *Workshop on Foundations of Deductive Databases and Logic Programming*, Washington, DC (Morgan-Kaufman, 1986) 19-88.
[33] M.H. Van Emdem and R.A. Kowalski, The semantics of predicate logic as a programming language, *J. ACM* **23**(4) (1976) 733-742.
[34] A. Van Gelder, Negation as failure using tight derivations for general logic programs, in: *Third IEEE Symp. on Logic Programming* (1986) 137-146.
[35] A. Van Gelder, K. Roth and J.S. Schlipf, Unfounded sets and well founded semantics for general logic programs, in: *Proc ACM SIGACT-SIGMOD Symp. on Principles of Database Systems* (1988) 221-230.