A Classification Theory of Semantics of Normal Logic Programs: II. Weak Properties

Jürgen Dix
University of Koblenz-Landau
Department of Computer Science
Rheinau 1,
56075 Koblenz, Germany

Abstract. Our aim in this article is to supplement the set of *strong* properties introduced in the preceding article ([Dix94]) with a set of *weak* principles in order to characterize semantics of logic programs. In [Dix94] we introduced our point of view: we observed that all semantics induce in a natural way a sceptical non-monotonic entailment relation SEM^{scept}. We ask for the properties of these sceptical relations and use them to describe all possible semantics. We collect in this paper serious shortcomings of some semantics proposed recently. Their strange behaviour led us to formulate in a natural way certain principles to avoid these problems. We argue that any well-behaved semantics should satisfy these principles. The main results state that our list of weak principles is complete in the following sense: any well-behaved-semantics is an extension of the well-founded semantics WFS and coincides for stratified programs with Apt, Blair, and Walker's supported model M_P^{supp}. We also claim that two extensions of the well-founded semantics (introduced in the preceding article) are uniquely characterized by their *strong* and *weak* properties.

1 Introduction

This article is the second in a series of three. The first two articles are devoted to *normal* programs while the third treats *disjunctive* programs. The methods and techniques (as well as some of the results) introduced here and in the first article are fundamental and will be extended in the third paper to disjunctive programs.

We refer the reader to the introduction of the first article ([Dix94]) where we gave some general remarks about the history of the problem. In Section 1.1 we present our own approach. The organization of the paper is given in Section 1.2.

1.1 Our approach

In this series of papers we intend to develop a framework which makes it possible to obtain results of the form

Any semantics satisfying certain properties is uniquely determined by these.

Having carefully investigated the recent approaches we discovered an irregular behaviour of some semantics. The reason is that one tried to improve a given semantics by putting an additional mechanism on top of its definition. This additional mechanism was often motivated by only one single program that, according to the intuitions of the respective group of researchers, was not handled correctly by the given semantics. We noticed that the new semantics sometimes have more serious shortcomings than the original semantics and therefore tried to find principles where all semantics should be checked against.

Our approach is partially inspired by the work of Kraus, Lehmann, Magidor and Makinson in general non-monotonic reasoning ([Mak89, Mak94, KLM90, DM92a]). They abstracted from particular (propositional versions of) non-monotonic logics, such as DL, AEL and CIRC and developed a general (*proof* and *model*) theory for non-monotonic relations " $\mid\!\sim$ " together with *soundness* and *completeness* results.

Following their approach, we will first associate to any semantics SEM a sceptical non-monotonic consequence relation SEM^{scept} and then axiomatically present two types of abstract properties of this relation:

- The first type, called *strong* principles, are adaptions of some of the properties introduced by Kraus, Lehmann, Magidor and Makinson: they have nothing to do with our special setting of logic programs but nevertheless will turn out to be useful. They have been investigated in the first article of this series and used to distinguish between some of the LP-semantics.

- The second type, called *weak* principles, reflect the specific idea of *negation-as-failure* in logic programming: the two clauses "$a \leftarrow \neg b$" and "$b \leftarrow \neg a$" are, viewed as *logic programs* completely different, but viewed as classical formulae, they are equivalent. The first clause states that b is false (because there is no clause with b in its head) and therefore a is true, while the second clause states that a is false and b true. These principles are defined and investigated in this paper. We argue that any semantics should be checked against these properties. In fact, all our properties were inspired by irregular behaviour of some of the existing semantics.

We claim that by taking both types of principles together, *weak* and *strong* properties can be used to uniquely characterize certain semantics. The corresponding representation conjectures will be stated and discussed in Section 5.7.

Figure 1 may help to illustrate the different classes of programs (together with semantics defined for them) that we are considering: while we are concerned in the first two articles with *normal* programs (the lower half of the diagram) we will extend our methods in the last article to disjunctive programs (the upper half of the diagram).

As already done in the preceding article, we refer the reader to three interesting overviews about negation in logic programming: Minker's article in the *Special Issue of the Journal of Logic Programming on Non-Monotonic Reasoning and Logic Programming* ([Min93]), Apt and Bol's article in the jubileum (10th anniversary) issue of the *Journal of Logic Programming*([AB94]) and the author's article in the *Proceedings of the Konstanz Colloquium in Logic and Information* ([Dix95]). While Minker's article gives an almost complete description of all the activities and different approaches in the field, Apt/Bol and Dix also try to present recent research results in a comprehensive and detailed manner. Although no proofs are given, all important definitions and notions are formally introduced to illustrate the underlying ideas in a precise and strict fashion. The articles of Apt/Bol and Dix can be seen as complementary in a sense: while Apt and Bol concentrate more on *LP-semantics*, Dix is more concerned with *NMR-semantics*.

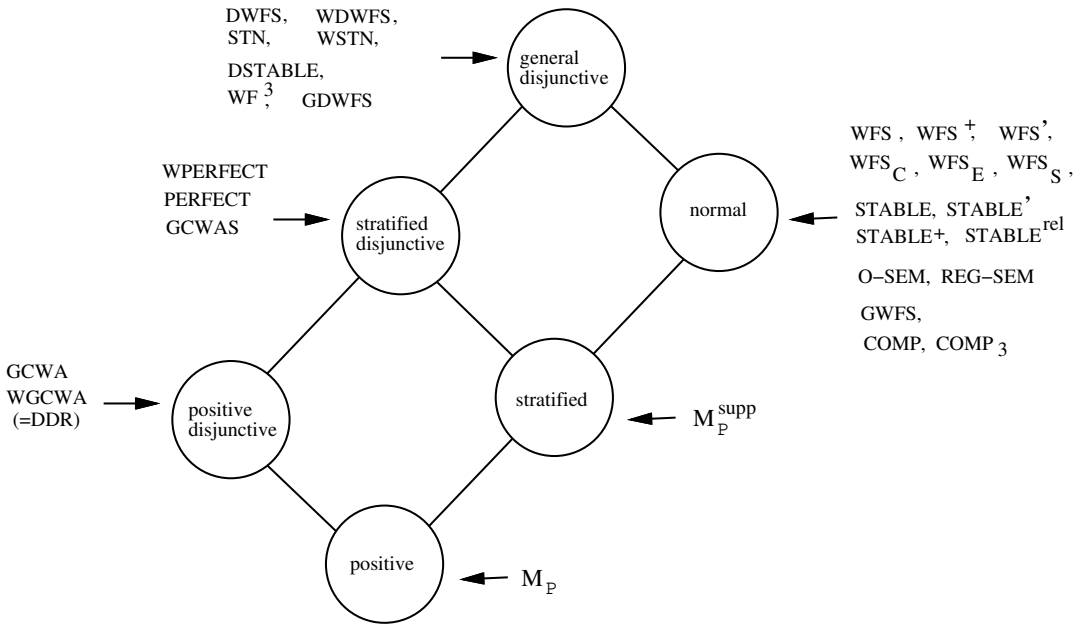


Figure 1: Classes of Programs

1.2 Organization of the paper

In the next section, Section 2, we briefly recall the standard terminology (introduced more detailed in the first article) that will be used in this series of papers. Section 3 recalls the by now classical extensions of M_P^{supp}: STABLE and WFS. We also state some of their properties needed in the sequel. Section 4 collects the most important extensions of the classical semantics and lists interesting examples to show their behaviour. These examples will be used in the next section to formulate our principles. Section 5 is the heart of the paper. Six principles are introduced, two theorems are proved and two interesting representation theorems are stated. Section 6 finally ends with some concluding remarks.

2 Notation and terminology

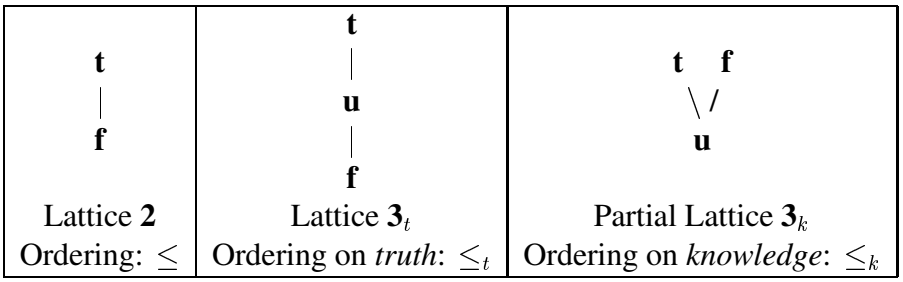
In Section 2.1 we recall some of the standard definitions for the context of logic programs, for our use of three-valued logic and for other notions used throughout the paper. In Section 2.2 we recall our definition of a semantics SEM, state the three most interesting rules of Kraus, Lehmann and Magidor and define the *sceptical entailment* SEM^{scept} of a semantics SEM. The complete discussion of all these notions is contained in the first article.

2.1 Logic programs and three-valued logic

A *general disjunctive* logic program consists of a finite number of *rules* that allow arbitrary *positive* clauses to appear in their heads:

$$A_1 \vee \ldots \vee A_n \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_l \quad \text{where } n \geq 1 \ .$$

If $l = 0$, the program is *positive disjunctive*; if $n = 1$ the program is *normal*; if $n = 1$ and $l = 0$ the program is *positive* (or *definite*) (see Figure 1).



Lattice **2**	Lattice $\mathbf{3}_t$	Partial Lattice $\mathbf{3}_k$
Ordering: $\leq$	Ordering on *truth*: $\leq_t$	Ordering on *knowledge*: $\leq_k$

Figure 2: Important (partial) lattices of truth-values

We denote the Herbrand base with respect to a program P by $B_{\mathcal{L}_P}$ or simply by B_P: the underlying language $\mathcal{L}_P$ is given by the symbols in P. Finally, let MIN-MOD(T) denote the class of all *two-valued* minimal Herbrand models of an arbitrary theory T. A Herbrand model $\mathcal{A}$ of T is called *minimal*, if there is no other model $\mathcal{A}'$ of T such that for all atoms a of the Herbrand base B_T: $\mathcal{A}' \models a$ implies $\mathcal{A} \models a$.

We also need some notions from 3-valued logic. We use truth values **t** "true", **f** "false", **u** "undefined" and the Kleene connectives $\vee, \wedge, \neg$ and $\leftarrow$. $\leftarrow$ is the *weak* implication, where "**u** $\leftarrow$ **u**" is considered to be *true*. Additionally, we can use two different orderings of the truth values as shown in Figure 2.

2.2 Sceptical semantics $\mathbf{SEM}_P^{scept}(U)$

In this article we consider the following NMR-semantics: the least Herbrand model M_P for *definite* programs, the supported Herbrand model M_P^{supp} ([ABW88]) for *stratified* programs and the following semantics defined for all programs: STABLE ([GL88, BF91b]), STABLE′, STABLE$^+$ (defined in Section 3.2 and Section 4.2), WFS ([vGRS88, vGRS91]), WFS$^+$ and WFS′ ([Dix92a]), WFS$_C$ ([Sch92]), WFS$_E$ ([CK91]), WFS$_S$ ([HY91]), GWFS ([BLM90]), O-SEM ([PAA92]) and the regular semantics REG-SEM ([YY90]) that was recently proved to be equivalent (see [YY93]) to Sacca and Zaniolo's *partial models* ([SZ91]), to Przymusinski's $\leq_k$*-maximal 3-valued stable models* ([Prz91, Prz90]) and to Dung's *preferred extensions* ([Dun91]). All these semantics are special instances of the following definition:

Definition 2.1 (SEM)
A semantics SEM is a mapping from the class of all programs into the powerset of the set of all 3-valued Herbrand structures. SEM assigns to every program P a set of 3-valued Herbrand models of P:

$$SEM_P \subseteq MOD_{3-val}^{Herb_{\mathcal{L}_P}}(P).$$

We will from now on use the notation $P \cup U$, where U is a set of *atoms* (no nontrivial program clauses are contained in U). P still may contain clauses with empty bodies. We also use the more terse $\mathrm{SEM}_P(U)$ instead of $\mathrm{SEM}_{P \cup U}$. This definition already indicates a fundamental difference to the general "$\sim$"-framework in non-monotonic reasoning: our $\sim_P$ is not defined between *arbitrary* program clauses.

The following structural properties for an entailment relation "$\sim$" *between single formulae* were considered by Kraus, Lehmann and Magidor:[1]

[1] "$\sim$" can be extended to a relation between *finite* sets of formulae using $\wedge$.

Cautious Monotony:	$\alpha \sim \beta$	and	$\alpha \sim \gamma$	$imply$	$\alpha \wedge \beta \sim \gamma.$
Cut:	$\alpha \sim \beta$	and	$\alpha \wedge \beta \sim \gamma$	$imply$	$\alpha \sim \gamma$
Rationality:	$not\ \alpha \nsim \neg\beta$	and	$\alpha \sim \gamma$	$imply$	$\alpha \wedge \beta \sim \gamma.$

We will now associate to any semantics SEM a sceptical entailment relation SEM^{scept} (also written $\sim_P$):

Definition 2.2 (Sceptical entailment relation $\sim_P$)
Let P be a program and U a set of atoms. Any semantics SEM induces a sceptical entailment relation SEM^{scept} as follows:

$$SEM_P^{scept}(U) := \bigcap_{\mathcal{M} \in \mathrm{SEM}_P(U)} \{L : L \text{ is a pos. or neg. literal with: } \mathcal{M} \models L\}$$

Comparing with the " $\sim$ " framework of Kraus, Lehmann and Magidor, we can equivalently define a " $\sim_P$"-relation between sets of atoms *U (positive literals) on the left hand side, and sets of arbitrary* literals *X on the right hand side:*

- $u_1 \wedge \ldots \wedge u_n \ \sim_P x_1 \wedge \ldots \wedge x_m$ **:iff** $\{x_1, \ldots, x_m\} \subseteq SEM_P^{scept}(\{u_1, \ldots, u_n\})$

Let us recall (this has been discussed in the first article) that we have to distinguish between two different notions of an *extension* of a semantics (by SEM we mean from now on SEM^{scept}):

- SEM $\leq_k$ SEM$'$: this means that SEM$'$ classifies more atoms as true or false than SEM, or
- SEM$'$ is defined for a class of programs that strictly includes the class of programs for which SEM is defined and for all programs of this smaller class, the two semantics coincide.

The first notion also makes perfectly sense for semantics defined for the same class of programs.

Like in the first article, all our results hold for *arbitrary* predicate logic programs. From now on, we will assume that a program P is just an abbreviation for its full instantiation: therefore, P stands for a (possibly infinite) *propositional* logic program. Some of our principles in Section 5 can be simplified for *finite* propositional programs (this point will be discussed below).

3 The classical semantics

We present in this section the classical approaches of defining semantics for logic programs. Section 3.1 shortly reviews the supported semantics M_P^{supp} for stratified programs: all NMR-semantics coincide with M_P^{supp} on this class of programs. But there exist also non-stratified programs which make perfectly sense: the stable semantics, presented in Section 3.2, and the well-founded semantics (Section 3.3) were constructed to give such programs a meaning:

Example 3.1 (Non-stratifiable program with unique semantics)

$$\begin{array}{lrcl} P_{default}: & v(a,b) & & \\ & v(b,c) & & \\ & e(c) & & \\ & e(x) & \leftarrow & v(x,y), \neg e(y) \end{array}$$

$P_{default}$ is not even locally stratifiable, because of $e(a) \leftarrow v(a,a), \neg e(a)$. Nevertheless it has an intuitive model:

$$\mathcal{M}_{P_{default}} = \{v(a,b), v(b,c), e(a), e(c)\}.$$

The clause $e(x) \leftarrow v(x,y), \neg e(y)$ can be interpreted as the description of a game, where the moving player wins, when his opponent has no more move to make[2]: the clause

- *wins(x) ← move_from_to(x,y), ¬ wins(y)*

has to be read as

- *the situation x is won (for the moving player A), if he can lead over[3] to a situation y that can never be won for B.*

Note that cycles in the relation move_from_to(,), or in v(x,y) respectively, make difficulties. Such cycles are handled differently by STABLE and WFS, as we will see below.

3.1 The supported semantics M_P^{supp}

The supported semantics is only defined for a subclass of the normal logic programs: the *stratified programs*. Let us consider

Example 3.2 (Positive and Negative Cycles)

$$P_{p \leftarrow p}: \quad p \leftarrow p, \qquad P_{p \leftarrow \neg p}: \quad p \leftarrow \neg p$$

In the first program, p depends positively on itself while in the second one, p depends negatively on itself. Therefore the only reason to assume p in the first program is p itself (hence p should not be derivable).

In the second program, the assumption $\neg p$ would imply a contradiction (hence $\neg p$ should not be derivable) but what should be assumed about p?

As we discussed in the first article, semantics related to SLDNF (like $comp_3$) assign p in the first program the value **u** (undefined). The problems with positive and negative cycles can be avoided for the class of *stratified* programs ([ABW88]) as defined in the first article. The idea is to rule out *all* programs having a cycle (not only a direct negative link) with a negative edge in their dependency graph. Programs without such cycles induce a natural priority ordering on their relation-symbols (see the preceding article [Dix94, Definition 3.6]). Przymusinski extended this construction to the class of *locally stratified* programs: he considered the infinite Herbrand base and not only the finite set of relation symbols. A program P is *locally stratified* if the obvious priority relation on B_P is noetherian[4] ([Prz88]). Like for stratified programs, a unique canonical two-valued Herbrand model can be constructed.

[2] As in checkers; see [vGRS88].

[3] With the help of a regular move, given by the relation $move_from_to(,)$.

[4] This means that there are no infinite descending chains.

3.2 STABLE

The idea of the stable semantics is that in an intended (two-valued) model any atom should have a definite reason to be true or false. This idea was made explicit in [BF91a] and, independently, [GL88]. We use the latter terminology and introduce the Gelfond-Lifschitz transformation: for a program P and a model $N \subseteq B_P$ we define

$$P^N := \{rule^N : \ rule \in P\}$$

where

$$(A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m)^N := \begin{cases} A \leftarrow B_1, \ldots, B_n, & \text{if } \forall j : C_j \notin N, \\ \mathbf{t}, & \text{otherwise.} \end{cases}$$

Note that P^N is always a *definite* program. We can therefore compute its least Herbrand model M_{P^N} and check whether it coincides with the model N with which we started:

Definition 3.3 (STABLE)
N is called a stable *model of P, if* $M_{P^N} = N$.

Let us refer to Example 3.1 and consider the program P consisting of the clause

$$wins(x) \ \leftarrow \ move_from_to(x, y), \ \neg wins(y)$$

together with the facts

$$move_from_to(a, b), \ move_from_to(b, a), \ move_from_to(b, c), \ move_from_to(c, d).$$

In this particular case we have two stable models: both contain $\{\neg wins(d), \ wins(c)\}$. In the first we also have $\{wins(b), \ \neg wins(a)\}$ while the second contains $\{\neg wins(b), \ wins(a)\}$. The sceptical semantics therefore derives $\neg wins(d)$ and $wins(c)$ which fits with our intuitions. Unfortunately, stable models do not always exist:

Example 3.4 ((Non-) existence of stable models)

$$\begin{array}{lcl} P_{\neg\exists\, stab.}: & a \leftarrow \neg b \\ & b \leftarrow \neg a \\ & p \leftarrow \neg p \end{array} \qquad \begin{array}{lcl} P_{stable}: & a \leftarrow \neg b \\ & b \leftarrow \neg a \\ & p \leftarrow \neg p \\ & p \leftarrow \neg a \end{array}$$

$P_{\neg\exists\, stab.}$ has no stable models. P_{stable} has the unique stable model $\{p, b\}$.

Example 3.5 (Adding irrelevant clauses)

$$P_{stratified}: \ a \leftarrow \neg b \qquad \begin{array}{lcl} P_{no_stable_model}: & a \leftarrow \neg b \\ & p \leftarrow \neg p \end{array}$$

The unique stable model of $P_{stratified}$ coincides with the supported model: therefore, a is derivable. If the clause $p \leftarrow \neg p$ is added, a is no longer derivable because no stable model exists.

The last example shows that the truth-value of an atom a also depends on atoms that are totally unrelated with a. One might think that this problem can be easily solved by redefining STABLE in the case where no stable models exist. The easiest way is to take the well-founded model WFS(P) and to define

$$\text{STABLE}^*(P) := \begin{cases} \text{WFS}(P), & \text{if no stable model exists,} \\ \text{STABLE}(P), & \text{otherwise.} \end{cases}$$

Unfortunately this only solves the problem with the two particular programs above. More complex programs, showing that even STABLE* suffers from the same shortcomings, can be easily constructed. We will come back to this point in Section 5.

The following programs are taken from [Dix91a] and show that STABLE is not cumulative (to be more precise, the *Cut* holds but *Cautious Monotony* fails):

Example 3.6 (STABLE is not cautious monotonic)

$$\begin{array}{lllll} P_{\neg cum}: & a \leftarrow \neg b & \qquad P_{Makinson}: & a \leftarrow \neg b \\ & b \leftarrow \neg a & & b \leftarrow \neg a, p \\ & p \leftarrow \neg p & & p \leftarrow a \\ & p \leftarrow a & & \end{array}$$

The first three clauses of $P_{\neg cum}$ have two minimal models $\mathcal{M}_1 = \{a, \neg b, p\}$ and $\mathcal{M}_2 = \{\neg a, b, p\}$, neither of them being stable. The fourth clause stabilizes $\mathcal{M}_1$*, i.e. the whole program has $\mathcal{M}_1$ as its only stable model. Therefore, STABLE($P_{\neg cum}$) implies p and a. But adding p to $P_{\neg cum}$, we get two stable models: $\mathcal{M}_1$ and $\mathcal{M}_2$. a does no longer follow from $P_{\neg cum} \cup \{p\}$! The same applies to $P_{Makinson}$.*

An interesting property of STABLE has been recently proved by Schlipf. This property can be seen as a *restricted* version of cumulativity:

Theorem 3.7 (Schlipf)
If $a \in$ WFS(P) then STABLE(P) = STABLE($P \cup \{a\}$).

3.3 WFS

The well-founded semantics has been introduced in the first article using Przymusinski's operator $\Phi_P^{\mathcal{J}}$. Unlike the stable models, the well-founded model is three-valued and always exists (but sometimes coincides with the empty model $\langle \emptyset; \emptyset \rangle$). For the example discussed in the last section

$$move_from_to(a,b),\ move_from_to(b,a),\ move_from_to(b,c),\ move_from_to(c,d),$$

and the clause $wins(x) \leftarrow move_from_to(x,y),\ \neg wins(y)$ the well-founded model is given by: $\{\neg wins(d),\ wins(c)\}$, thus

$$\text{WFS}(P) = \bigcap_{\mathcal{N} \text{ a stable model of } P} \mathcal{N}.$$

But this is not always the case, as the program $P_{splitting\,I}$ defined in the next section will show. In general we have the following close relationship between WFS and STABLE ([vGRS91]):

- Every stable model $\mathcal{N}$ of P is an extension of WFS(P): WFS(P) $\leq_k \mathcal{N}$.

- If WFS(P) is two-valued, WFS(P) is the unique stable model.

The main difference between WFS and STABLE is that in the definition of the former, more and more atoms are declared to be true (or false): once a decision has been drawn, it will never be rejected. In the definition of STABLE however, a *guess* is made and then a particular model is constructed and used to justify the guess or to reject it. We will now define a notion of

"reducing a program using a set of literals M"

that corresponds more closely to the spirit of the first construction. Nevertheless, this notion is not related to a particular semantics. It does not formalize a particular reasoning mode. It will be used in Section 5 and help us to formulate some interesting conditions.

Definition 3.8 (P reduced by M)
Let P be a program and M be a set of literals. "P reduced by M" is the program

$$P^M := \{rule^M : rule \in P\},$$

where $(A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m)^M$ is defined by

$$(A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m)^M := \begin{cases} \mathbf{t}, & \text{if } \exists j : C_j \in M \text{ or } \neg B_j \in M, \\ \mathbf{t}, & \text{if } A \in M \text{ or } \neg A \in M, \\ rule', & \text{otherwise.} \end{cases}$$

Here, $rule'$ stands for the clause "$A \leftarrow B'_1, \ldots, B'_{n'}, \neg C'_1, \ldots, \neg C'_{m'}$", where the set $\{B'_i : i \in I'\}$ (resp. $\{\neg C'_i : i \in I'\}$) is just an enumeration of the set $\{B_i : i \in I\} \setminus M$ (resp. $\{\neg C_i : i \in I\} \setminus M$).

We define the associated language $\mathcal{L}_{P^M}$ for P^M to be $\mathcal{L}_P \setminus M$, i.e. $\mathcal{L}_{P^M}$ consists of all symbols occurring in P but different from those in M.[5]

This notion of reduction resembles the Gelfond-Lifschitz transformation to define stable models but it will be used totally differently:

- Our reduced program still contains negation: the Gelfond-Lifschitz transformation is a positive program.
- While Gelfond/Lifschitz reduce with a set N of *atoms* and also implicitly assume that all atoms not in N are false, we are more cautious and reduce only those literals that are explicitly contained in M. We *remove* all atoms that are already known to be true or false: once **t** or **f** is assigned, it is fixed. We use this information to simplify the program: our reduction decreases the Herbrand base (see [MD93]). Literals contained in M do no longer occur in P!

The reduction P^M will lead us to formulate a *closure condition* that expresses the fact that if one reduces a program by its own semantics (i.e. we build $P^{SEM(P)}$) then a further application of SEM to this program should not yield anything new: $SEM(P^{SEM(P)}) = \emptyset$. We will discuss this property and its strong relation to the Cut in Section 5.

Using our reduction, a detailed investigation of the constructions in [Prz92] gives the following equivalent formulation of WFS[6]:

[5] The careful reader notes that this is in general a superset of the symbols occurring in P^M.

[6] The lemma evolved from discussions with R. Bol and Th. Fuchß.

Lemma 3.9 (WFS)
Let P be an instantiated program, P^+ the program obtained from P by cancelling all negative literals, and let P^- be the program obtained from P by cancelling all program-clauses containing a negative literal. Note that P^+ as well as P^- are definite programs with associated least Herbrand model.

Let $P_0 := P$, $M_0^+ = M_0^- = \emptyset$ and for successor ordinals $\alpha + 1$

- $M^-_{\alpha+1} = \{\neg x : x \in \mathcal{L}_{P_\alpha}$ *and* $x \notin M_{P_\alpha \setminus \neg B_{P_\alpha}}\}$,
- $M^+_{\alpha+1} = \{x :\ M_{P^-_\alpha} \models x\}$ *and*
- $P_{\alpha+1} := P_\alpha^{M^-_\alpha \cup M^+_\alpha}$.

For limit ordinals λ we define $M^\pm_\lambda = \bigcup_{\alpha \prec \lambda} M_\alpha$ and $P_\lambda = P^{M^-_\lambda \cup M^+_\lambda}$.

Note that $\mathcal{L}_{P_{\alpha+1}} \subseteq \mathcal{L}_{P_\alpha}$. The construction ends if $\mathcal{L}_{P_{\gamma+1}} = \mathcal{L}_{P_\gamma}$. We have

$$\mathit{WFS}(P) = \bigcup_{\alpha=0}^{\lambda} M^+_\alpha \ \cup \ \bigcup_{\alpha=0}^{\lambda} M^-_\alpha$$

Our remark about the underlying language in Definition 3.8 is very important. Let us consider the program "$a \leftarrow \neg b$, $b \leftarrow \neg c$". In the first step we have $P_0^+ = \{a \leftarrow , b \leftarrow\}$, $P_0^- = \emptyset$ and

$$M_{P_0^+} = \{a, b\}, \ \ M_{P_0^-} = \emptyset.$$

We have therefore to reduce P with $\neg c$ and get $P_1 = \{a \leftarrow \neg b,\ b \leftarrow\}$. Then we get $M_1^- = \emptyset$, $M_1^+ = \{b\}$ and $P_2 = \emptyset$ **with respect to** $\mathcal{L}_{P_2} = \{a\}$, i.e.

$$M_2^- = \{\neg a\}.$$

The construction ends and we have $\{\neg c, b, \neg a\}$ as expected.

4 Extensions of WFS

The following example is representative for the problems with *reasoning by cases* or *case-splitting*. Considering only two-valued models, some researchers argue that p should be derivable in the following

Example 4.1 (Case-splitting I)

$$\begin{array}{lrcl} P_{splitting\,I}: & a & \leftarrow & \neg b \\ & b & \leftarrow & \neg a \\ & p & \leftarrow & a \\ & p & \leftarrow & b \end{array}$$

Although neither a nor b can be "derived" in any semantics based on two-valued models (as STABLE for example) the disjunction $a \vee b$, thus also p, is true. In this way the example is handled by the completion-semantics too. Baral, Lobo and Minker argue ([BLM90]) that p should be derivable. WFS(P) does not fulfill this.

The simplest way to augment WFS for this task is by using minimal model reasoning. There are, however, various possibilities.

4.1 GWFS, EWFS

The simplest one is first evaluating WFS and then adding all literals that are true in all minimal models extending WFS(P): this defines EWFS, a precursor of WFS$^+$ defined and discussed in the preceding article. Now, this procedure may add new literals in such a way that a further application of WFS still yields new literals: *Cut* is not satisfied. In order to get some *"closure"* of this process, *Baral, Lobo and Minker* defined GWFS (see [BLM90]).

The exact realization of the intuitive idea of extending WFS by adding all literals true in all minimal models extending WFS is by using the operator

$$\mathbf{\Omega}_P : {\mathbf{3}_k}^{B_P} \longrightarrow {\mathbf{3}_k}^{B_P}; \mathcal{J} \longmapsto \mathbf{\Phi}_P^{\mathcal{J}}\uparrow^{\omega},$$

which is based on the definition of

$$\mathbf{\Phi}_P^{\mathcal{J}} : {\mathbf{3}_t}^{B_P} \longrightarrow {\mathbf{3}_t}^{B_P}; \mathcal{I} \longmapsto \mathbf{\Phi}_P^{\mathcal{J}}(\mathcal{I})$$

(see Section 4.1 of [Dix94]). Note that WFS(P):= $lfp(\mathbf{\Omega}_P)$. GWFS is then defined as follows:

Definition 4.2 (GWFS, T($\mathcal{J}$), F($\mathcal{J}$))
For a three-valued interpretation $\mathcal{J}$, let $\mathcal{J}-$MIN-MOD(P) denote the class of all minimal two-valued Herbrand models of P that are consistent with $\mathcal{J}$. Furthermore, let

- **T**$(\mathcal{J})$:= *True($\mathcal{J}-$MIN-MOD(P)) and* **F**$(\mathcal{J})$:= *False($\mathcal{J}-$MIN-MOD(P)).*

Here, for a set $\mathcal{S}$ of Herbrand models, True($\mathcal{S}$) (resp. False($\mathcal{S}$)) stands for the set of all ground atoms A, which are true (resp. the negations of which are true) in all models in $\mathcal{S}$[7]*.*

We define

$$\textit{GWFS(P):}= lfp(\mathbf{\Omega}_P^G),$$

where the operator $\mathbf{\Omega}_P^G$ is defined as follows:

$$\mathbf{\Omega}_P^G : {\mathbf{3}_k}^{B_P} \longmapsto {\mathbf{3}_k}^{B_P}; \mathcal{J} \longmapsto \mathbf{\Phi}_P^{\mathcal{J}}\uparrow^{\omega} + \langle \mathbf{T}(\mathcal{J}), \mathbf{F}(\mathcal{J}) \rangle.$$

While *Cut* is satisfied by GWFS, this semantics shows another serious shortcoming:

Example 4.3 (Strange behaviour of GWFS)

$$P_{GWFS}: \begin{array}{lcl} p & \leftarrow & \neg b \\ b & \leftarrow & c \\ c & \leftarrow & p, \neg a \\ a & \leftarrow & \neg b \end{array} \qquad P_{GWFSc}: \begin{array}{lcl} p & \leftarrow & \neg b \\ b & \leftarrow & p, \neg a \\ a & \leftarrow & \neg b \end{array}$$

GWFS(P_{GWFS}) entails $\neg c$, because MIN-MOD(P_{GWFS})=$\{ \{p, a\}, \{b\} \}$ and thus also $\neg b$, p and a. But GWFS(P_{GWFSc}) neither implies p nor a.

Note that intuitively, P_{GWFS} can be seen as an *extension-by-definition* of P_{GWFSc}: c is nothing than just an abbreviation (a definition) for $p \wedge \neg a$!

[7] If $\mathcal{S} = \emptyset$, we define True($\mathcal{S}$) (resp. False($\mathcal{S}$)) to be equal to $\emptyset$.

4.2 $\mathbf{WFS}_C$, $\mathbf{WFS}_E$, $\mathbf{WFS}_S$

In the recent article [Sch92], Schlipf collected some "common-sense goals" for negation-as-failure and noticed that none of the existing semantics achieves them all. He constructs an extension of the well-founded semantics, the well-founded-by-case-semantics WFS_C and proves that this semantics satisfies all the goals except the *GCWH-property*:

- *GCWH-property*: $\text{MIN-MOD}(P) \models \neg a \implies \neg a \in \text{SEM}(P)$.

We do not think that this property is a good one: it is strongly related to the phenomenon in Example 4.3, where it is the reason that GWFS does not respect *extensions-by-definition*.

Modifying Schlipf's construction, Hu and Yuan define in [HY91] another extension WFS_E of WFS_C. The difference may be illustrated with the following

Example 4.4 (Case-splitting II)

$$P_{splitting\,II}: \quad \begin{array}{lll} a & \leftarrow & \neg b \\ b & \leftarrow & \neg a \\ c & \leftarrow & \neg a, \neg b \end{array}$$

$\text{WFS}_C(P_{splitting\,II}) = \emptyset$ but $\text{WFS}_E(P_{splitting\,II}) = \{\neg c\}$.

Concerning simple negative loops ($a \leftarrow \neg a$) the behaviour of WFS_E is similar to WFS^+: a is evaluated to *true*. But the *extended Cut* is not satisfied:

$$\text{WFS}_E(P_{GWFS}) = \{\neg c\}.$$

Adding $\neg c$ (as defined at the end of Section 2.3 of the first article) makes $\neg b$ derivable.

Still another extension of WFS_C is defined by Chen and Kundu in [CK91]: WFS_S, the *strong* well-founded semantics. WFS_S has the GCWH-property. Again, the *extended Cut* is not satisfied.

We have just illustrated that both semantics WFS_E and WFS_S do not satisfy the *extended Cut*. We gave a similar example of the failure of *extended Cut* for WFS^+ in the first article (following the Remark 2.7). But in contrast to WFS^+, the semantics WFS_E and WFS_S do not even satisfy the closure condition $SEM(P^{SEM(P)}) = \emptyset$ (the examples just given are also counterexamples for this property).

More generally we can show that *any* semantics respecting our principle of *extension-by-definition* (later called weak PPE), the Closure-condition the *Cut*, and the GCWH-property is already determined on the following three programs:

Example 4.5 (Behaviour of GCWH-property)

$$P_{GCWH}: \quad \begin{array}{lll} a & \leftarrow & \neg b \\ b & \leftarrow & \neg b, x \\ x & \leftarrow & \neg a, \neg b \\ x & \leftarrow & a \end{array} \qquad P_{GCWH_a}: \quad \begin{array}{lll} b & \leftarrow & \neg b, x \\ b & \leftarrow & \neg a, \neg b \\ x & \leftarrow & \neg b \end{array} \qquad P_{GCWH_{ax}}: \quad b \leftarrow \neg b$$

Since MIN-MOD$(P) = \{\{b\}\}$ we have $\neg a, \neg b \in$ SEM(P). We also have $P^{\{\neg a, \neg\}} = \emptyset$ with respect to the language containing b, i.e. SEM$(P^{\{\neg a, \neg\}}) = \{\neg b\}$ so that the Closure condition is not satisfied. Therefore either $\neg b \in$ SEM(P) or $b \in$ SEM(P). The first possibility would imply $a \in$ SEM(P) which is a contradiction (SEM(P) should be a model of P). So we have: SEM$(P_{GCWH}) = \{\neg a, b, \neg x\}$.

Applying our extension-by-definition principle to P_{GCWH} and replacing a we get P_{GCWH_a}. Replacing now x and simplifying, we get $P_{GCWH_{ax}}$. As we regard all our transformations equivalence-preserving, we have proved that SEM$(b \leftarrow \neg b) = \{b\}$.

The GCWH-principle therefore seems to be very strong.

In the first article we constructed an extension WFS$^+$ of WFS that satisfies *Supraclassicality* (if $P \models A$ then $A \in WFS^+$(P)). Obviously, supraclassical semantics derive p in programs containing the clause $p \leftarrow \neg p$ (classically, this clause is equivalent to p). One might think therefore that a supraclassical extension of STABLE removes the inconsistency problem. Schlipf defined in [Sch92] to this end not only a supraclassical extension of WFS (which we prove below to be identical with WFS$^+$) but also a supraclassical version of STABLE: STABLE$_C$ (STABLE-BY-CASE). Both constructions are technical and complicated. Using our abstract properties we can give a very elegant definition of an improvement of STABLE that satisfies *Supraclassicality* (analogously to Theorem 4.9 of the first article where we proved that WFS$^+$ is the smallest supraclassical extension of WFS satisfying *Cut*). Note that we cannot use Theorem 4.6 of the first article because STABLE is not rational. Fortunately it turns out that things are much easier:

Definition 4.6 (STABLE$'$)

$$\textit{STABLE}'(P) := \begin{cases} \textit{STABLE}(P \cup \{a : \ P \models a\}), & \textit{if stable models exist,} \\ \textit{WFS}^+(P), & \textit{otherwise.} \end{cases}$$

Lemma 4.7 (STABLE$'$ $\leq_k$ STABLE)
STABLE$'$ is a supraclassical semantics below STABLE satisfying Cut.

Proof: We only have to prove that STABLE$'$ satisfies the *Cut*. Let $\beta \in$ STABLE$'(P)$ and $\gamma \in$ STABLE$'(P \cup \{\beta\})$. We have to show that $\gamma \in$ STABLE$'(P)$. This is trivially true if STABLE$'(P) = WFS^+(P)$. Let us therefore assume that there exists at least one stable model of $P \cup \{a : \ P \models a\}$. Our assumption about β (i.e. every stable model of $P \cup \{a : \ P \models a\}$ satisfies β) and the fact that *Cut* holds for STABLE gives:

$$\gamma \in \text{STABLE}(P \cup \{\beta\} \cup \{a : \ P \models a\}) \text{ implies } \gamma \in \text{STABLE}(P \cup \{a : \ P \models a\}).$$

Our assumption about γ is $\gamma \in$ STABLE$(P \cup \{\beta\} \cup \{a : \ P \cup \{\beta\} \models a\})$. But we also have[8]

$$\{a : \ P \cup \{\beta\} \models a\} \subseteq \text{STABLE}(P \cup \{a : \ P \models a\})$$

so we can use *Cut* again and get $\gamma \in$ STABLE$(P \cup \{\beta\} \cup \{a : \ P \models a\})$. ∎

Why is STABLE$'$ weaker than STABLE? This is just because STABLE is not cumulative and some inconsistencies are removed: for the program "$p \leftarrow \neg p$" STABLE is inconsistent while STABLE$'$ is not. The introduction of new atoms in general *decreases* the number of stable models.

Note that while STABLE$_C$ might still be inconsistent for some programs P, our STABLE$'$ is always consistent (by construction). Schlipf's STABLE$_C$ is still weaker than STABLE$'$: we refer the reader to [DM94c] where we comment on this and where we defined (beside other things) a semantics STABLE$^+$ that coincides with the consistent part of STABLE$_C$.

We will end this section with the observation that, although very differently defined, WFS$_C$ and WFS$^+$ are identical:

[8]Here we use essentially that stable models are two-valued, so that all classical consequences of $P \cup \{\beta\}$ hold in all stable models. For three-valued stable models (i.e. *justified* models as defined in Definition 4.10 of the previous article) this does not hold.

Theorem 4.8 (WFS$^+$=WFS$_C$)
For all programs P: WFS$^+(P)$ = WFS$_C(P)$.

Proof: We do not present the (complicated) definition of WFS$_C(P)$ (the reader is referred to [Sch92]) and only sketch the proof. We use the definition of WFS$^+$ ([Dix94, Definition 4.8]).

Using [Sch92, Theorem 5.1] it is straightforward to prove

$$\text{WFS}_C(P \cup M_\gamma) = \text{WFS}_C(P).$$

To prove

$$\text{WFS}(P \cup M_\gamma) \leq_k \text{WFS}_C(P)$$

by induction on γ is done by using [Sch92, Theorem 5.9]:

$$\text{WFS}(P \cup M_\gamma) \leq_k \text{WFS}_C(P \cup M_\gamma) = \text{WFS}_C(P).$$

Therefore it follows WFS$^+(P) \leq_k$ WFS$_C(P)$.

The opposite direction can be seen as follows. Suppose an atom a is in WFS$_C(P)$. This means that a is true in all three-valued models of the stable-by-case completion of P. This can only happen if some of the q_{ij} in

$$a \leftrightarrow [(\neg q_{1,1} \wedge \ldots \wedge \neg q_{1,n_1}) \vee \cdots \vee (\neg q_{n,1} \wedge \ldots \wedge \neg q_{n,n_n})]$$

evaluate to "false" in such a way that the whole conjunct becomes "true". But then we can trace back all q_i to q_i' such that "$q_i' \Leftarrow$ " is a derived rule and therefore ([Sch92, Theorem 5.1]) $P \models q_i'$. The derivation can then be simulated within WFS$^+$. The dual applies to $\neg a$. ∎

4.3 The relationship between the semantics

The following program is due to J. Minker. Carolina Ruiz pointed out that it shows the incomparability of STABLE (and of STABLE^{rel} defined in [DM94b]) and GWFS:

$$\begin{array}{lrcl} P: & a & \leftarrow & \neg b \\ & b & \leftarrow & \neg a \\ & p & \leftarrow & a, \neg c \\ & p & \leftarrow & b \\ & c & \leftarrow & \neg d \\ & d & \leftarrow & a \end{array}$$

P has two stable models: $\{a, d, p\}$ and $\{b, c, p\}$ so that p is derivable. Using GWFS however, p is not derivable since there is also the minimal model $\{a, d, c\}$.

Note that depending on the interpretation of the nonexistence of stable models, WFS and STABLE might be incomparable with respect to our relation $\leq_k$: in Example 3.5 we can derive a from $P_{no_stable_model}$ using WFS but not using STABLE. Using our modification STABLE* however, we get a more regular behaviour: WFS$(P) \leq_k$ STABLE$^*(P)$. But we can also view the inconsistency of STABLE as *deriving anything*: under this interpretation we have

$$\text{WFS}(P) \leq_k \text{STABLE}^*(P) \leq_k \text{STABLE}(P).$$

The program $P_{splitting\,I}$ shows that STABLE $\not\leq_k$ WFS. Note that while it is not the case that WFS$_C$ $\leq_k$ STABLE, it is true that any stable model of P (if it exists) is an extension of WFS$_C(P)$ (viewed as a three-valued model).

$P_{\neg cum}$ from Example 3.6 also shows that REG-SEM (see [Dix94, Definition 4.10]) is weaker than STABLE: REG-SEM$(P_{\neg cum})$ = WFS$(P_{\neg cum})$ = $\emptyset$. This is because both $\langle\{a,p\};\{b\}\rangle$ and $\langle\{b\};\{a\}\rangle$ are regular models of $P_{\neg cum}$. Therefore:

$$\text{WFS} \leq_k \text{WFS}' \leq_k \text{REG-SEM} \leq_k \text{STABLE}.$$

However, REG-SEM is incomparable with STABLE$_C$: REG-SEM derives x from the program "$a \leftarrow \neg b$, $b \leftarrow \neg a$, $x \leftarrow \neg a$, $x \leftarrow \neg b$" while STABLE$_C$ does not.

Let us again note that REG-SEM, the semantics based on regular models ([YY90]) is representative for the following semantics (proved to be equivalent by You and Yuan ([YY93])

- Sacca and Zaniolo's *partial models* ([SZ91]), to
- Przymusinski's $\leq_k$*-maximal 3-valued stable models* ([Prz91, Prz90]), and
- Dung's *preferred extensions* ([Dun91]).

All these relations are illustrated in Figure 4 and Figure 5.

5 Well-behaved semantics

In this section we list nine principles that should hold for any semantics whatsoever. Section 5.1 presents the first two conditions: *Reduction* and *Relevance*. *Reduction* formalizes that *explicit facts* (resp. *explicit missing predicates*) should be considered true (resp. false). *Relevance* is induced from the strange behaviour of STABLE. Section 5.2 presents variants of *PPE* and *Modularity*: properties induced from irregular behaviour of GWFS and STN (a semantics for disjunctive programs to be discussed in the third article of this series). In Section 5.3 we discuss the *Cut* and some instances of the *extended Cut* that can be viewed as *closure conditions*: they are induced from irregular behaviour of WFS$_E$ and WFS$_S$. The next two principles (M_P*-extension* and *Transformation* discussed in Section 5.4) are abstracted from the supported semantics and the underlying NMR-intuition. Section 5.5 illustrates the relationship between some of these conditions. For finite propositional programs, some conditions are obsolete. We define in Section 5.6 the notion of a *well-behaved* semantics and prove two important theorems. In Section 5.7, finally, we formulate two representation-conjectures.

We already emphasized the fact that all known semantics (see Figure 4) are defined by declaring some of the (three-valued) models of P as the intended models. It turns out that the supported model, the stable models (which are two-valued) and also the well-founded model satisfy the following conditions (this was noted by L. Pereira and explicitly stated in [PAA92]): (let $\mathcal{A}$ be such a model)

C$_1$: If $a \leftarrow rhs \in P$ and $\mathcal{A} \models rhs$, then $\mathcal{A} \models a$.

C$_2$: If $\mathcal{A} \models a$ then there is $a \leftarrow rhs \in P$ with $\mathcal{A} \models rhs$.

C$_3$: If all rule bodies for a are false in $\mathcal{A}$ then $\mathcal{A} \models \neg a$.

C$_4$: If $\mathcal{A} \models \neg a$ then all rule bodies for a are false in $\mathcal{A}$.

Note that we are always interested in the associated sceptical semantics SEM^{scept} which is given by the $\sqcap_k$-intersection of all intended models. We want to list reasonable properties of SEM^{scept} in order to describe particular semantics as uniquely determined by these properties. Any sceptical semantics SEM^{scept} such that SEM is based on models satisfying C_1 - C_4, also satisfies C_1 and C_3 but not necessarily C_2 and C_4. We only consider C_1 as a property that should be always satisfied. WFS$^+$ (and also GWFS) do not satisfy C_3 because WFS$^+(\{a \leftarrow \neg a\}) = \{a\}$. WFS′ however satisfies C_3 (see [Dix94, Theorem 4.13]).

5.1 Relevance and Reduction

To state the following two conditions, we need the notions of the dependency-graph $\mathcal{G}_P$ and of P reduced by M as defined in Section 3.3. The condition *Relevance* also uses the two notions:

- $dependencies_of(X) := \{A : X \text{ depends on } A\}$, and
- $rel_rul(P, X)$ is the set of *relevant rules* of P with respect to X, i.e. the set of rules that contain an $A \in dependencies_of(X)$ in their heads.

Given any semantics SEM and a program P, it is perfectly reasonable that the truth-value of a literal L, with respect to SEM(P), only depends on the subprogram formed from the *relevant rules* of P with respect to L.[9] This idea is formalized by:

Definition 5.1 (Relevance)
Relevance states that for all literals L: SEM$(P)(L) =$ SEM$(rel_rul(P, L))(L)$.

Note that the set of relevant rules of a program P with respect to a literal L contains all rules, that could ever contribute to L's derivation (or to its nonderivability). In general, L depends on a large set of atoms: $dependencies_of(L) := \{A : L \text{ depends on } A\}$. But rules that do not contain these atoms in their heads will never contribute to their derivation or non-derivation. Therefore, these rules should not affect the meaning of L in P.

STABLE does not satisfy this principle. This is not only due to the nonexistence of stable models after adding a clause "$c \leftarrow \neg c$" to a program (see Example 3.5), it is in a sense intrinsic in the definition of stable models: $P_{\neg cum}$ from Example 3.6 has a consistent stable semantics, but

$$\text{STABLE}(P_{\neg cum})(a) = \mathbf{t} \neq \mathbf{u} = \text{STABLE}(rel_rul(P_{\neg cum}, a))(a).$$

Although our modified definition STABLE* in Section 3.2 seems to be very natural it also does not satisfy *Relevance*. The following semantics

$$\text{STABLE}^{\#}(P)(L) := \begin{cases} \text{STABLE}(rel_rul(P, L))(L), & \text{if there exist stable models ,} \\ \text{WFS}(rel_rul(P, L))(L), & \text{otherwise,} \end{cases}$$

satisfies *Relevance* but, unfortunately does not satisfy *Cut*. Indeed, it is not even a model of the underlying program. The interested reader is referred to [DM94b] where we construct a version of STABLE that satisfies all our weak properties (see Lemma 5.31).

There is another natural property satisfied for all semantics in the literature. Suppose we add a set N of atoms to a program P. No matter whether some of the atoms or their negations are derivable from P alone or not, the semantics of $P \cup N$ should be the same as the semantics of "P reduced by N" (see Definition 3.8), augmented by N. This idea can be generalized in two directions:

[9] Let $dependencies_of(\neg X) := dependencies_of(X)$, and $rel_rul(P, \neg X) := rel_rul(P, X)$.

1. We can allow N to contain negative literals. We define P to be equal to $P \cup \{\neg A\}$ just in case A does not appear in any head of P.

2. We can allow to replace any positive occurrence of an atom A by the associated defining rules.

The formal statement for possibility 1. is

Definition 5.2 (Reduction)
Let a consistent set of literals $M \subseteq B_P \cup \neg B_P$ be given.
The principle of Reduction states that SEM$(P \cup M)$ = SEM$(P^M) \cup M$.

It is important to take a closer look at this property and to make sure that it is indeed a very weak property. It simply says that whenever we have an explicit fact "$a \leftarrow$ " then we can replace every occurrence of a by "**t**" (dually: every occurrence of $\neg a$ by "**f**"). But note also our convention of using "$P \cup \{\neg x\}$" (see the end of Section 2.3 in the first article): "$P \cup \{\neg x\}$" is just an abbreviation for the program obtained from P by deleting all clauses with x in their heads and keeping the language of P ($\mathcal{L}_{P \cup \{\neg x\}} := \mathcal{L}_P$). Therefore *Reduction* assures that atoms that do not occur in the heads of a program are always assigned "**f**". This property should definitely be among the minimal requirements for semantics of logic programs: it is nothing else than a very weak form of the *negation-as-failure*-idea.

The formal statement for possibility 2. is the *Principle of Partial Evaluation* introduced in the next section.

5.2 PPE and Modularity

In Example 4.3 we pointed out a strange behaviour of GWFS. Our principle of partial evaluation, PPE, states that this should not happen: any semantics should assign the same meaning to a program P and a partial evaluation of it.

The formal definition is as follows:

Definition 5.3 (PPE, weak version)
Let P be an instantiated program and let an atom c occur only positively in P. Let $c \leftarrow rhs_1, \ \ldots \ , c \leftarrow rhs_n$ be all the rules of P with c in their heads. Assume further that $rhs_1, \ldots, rhs_n$ do not contain c.

We denote by P_c the program obtained from P by deleting all rules with c in their heads and replacing each rule "$head \leftarrow c, body$" containing c by the rules:

$$\begin{array}{lcl} head & \leftarrow & rhs_1, body \\ & \vdots & \\ head & \leftarrow & rhs_n, body \end{array}$$

The weak principle of partial evaluation *PPE states that*

$$\textit{SEM}(P_c) = \textit{SEM}(P) \setminus \{c, \neg c\}.$$

PPE can significantly decrease the complexity of computing a semantics, since it allows us to *reduce* the program by making the underlying Herbrand base smaller: in general, less assignments of elements of B_P have to be taken into account (see [DM93]).

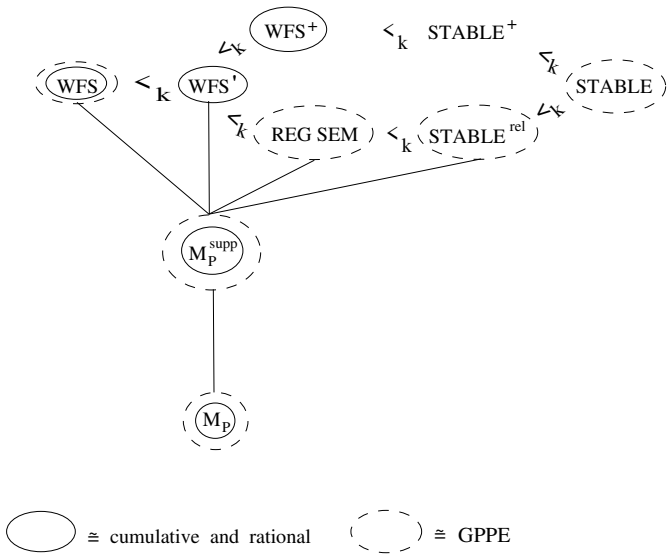


Figure 3: Semantics wrt Rationality and GPPE

Definition 5.4 (PPE)
Let P be an instantiated program and let an atom c occur only positively in P. Let $c \leftarrow rhs_1, \ldots, c \leftarrow rhs_n$ be all the rules of P with c in their heads.

Any program clause of the form "$head \leftarrow c, body$" can be replaced by the rules

$$\begin{array}{rcl} head & \leftarrow & rhs_1, body \\ & \vdots & \\ head & \leftarrow & rhs_n, body \end{array}$$

Note that the rules $c \leftarrow rhs_1, \ldots, c \leftarrow rhs_n$ are not removed (in contrast to the weak version of PPE). We call the program obtained in this way P'.

The principle of partial evaluation is: SEM(P') = SEM(P).

We obtain a much stronger version of this principle when we cancel the assumption about c:

Definition 5.5 (GPPE)
GPPE is obtained from PPE by allowing c to occur arbitrarily in P.

This variant is the general *partial deduction technique* known in Logic Programming. It was introduced in [Dix95, DM94b] and for general disjunctive programs independently in [BD94c] and [SS94]. See [DM94b] for a discussion of this property concerning various semantics (notably WFS, STABLE and REG-SEM satisfy it): Figure 3 gives an overview.

The next principle, *Modularity*, has some similarities with PPE. It enables us to compute a semantics by modularizing it into certain "subprograms" (formed of the relevant rules). The semantics of these modules can be computed first and the semantics of the whole program can be determined by reducing this program with literals that were already determined.

Definition 5.6 (Weak Modularity)
Let P be instantiated, $P = P_1 \cup P_2$, $B_{P_1} \cap B_{P_2} = \{A\}$ and $P_2 = rel_rul(P, A)$.
Then: SEM(P) = SEM($P_1{}^{SEM(P_2)} \cup P_2$).

We showed in [Dix92a] that the stationary approach of Przymusinski for disjunctive programs does not satisfy Weak Modularity. We do not know of any semantics for normal programs that fails to satisfy Weak Modularity (in the case of normal programs, Przymusinski`s approach coincides with WFS). It is useful to strengthen this condition by weakening the assumptions "$B_{P_1} \cap B_{P_2} = \{A\}$ and $P_2 = rel_rul(P, A)$":

Definition 5.7 (Modularity)
Let P be instantiated, $P = P_1 \cup P_2$ *and for every* $A \in B_{P_2}$: $rel_rul(P, A) \subseteq P_2$.
The principle of Modularity is: SEM(P) = SEM$({P_1}^{SEM(P_2)} \cup P_2)$.

Note that in the last definition it is **not** possible to replace the condition

$$\text{SEM}(P) = \text{SEM}({P_1}^{SEM(P_2)} \cup P_2)$$

by

$$\text{SEM}(P) = \text{SEM}({P_1}^{SEM(P_2)}) \cup \text{SEM}(P_2)$$

or by

$$\text{SEM}(P) = \text{SEM}({P_1}^{SEM(P_2)} \cup \text{SEM}(P_2)).$$

This would exclude floating-conclusion phenomena: choose $P_1 = \{x \leftarrow a,\ x \leftarrow b\}$ and $P_2 = \{a \leftarrow \neg b,\ b \leftarrow \neg a\}$. The modified definition would not allow us to deduce x because SEM$(P_2) = \emptyset$ and therefore SEM(P) = SEM$(P_1) \not\supseteq \{x\}$ (for both variants). But for our first version of Modularity, Weak Modularity, this modification makes sense and follows from our principles: see Section 5.5.

5.3 Closure Conditions

We already formulated after Definition 3.8 a closure condition:

Definition 5.8 (Closure)
The principle of Closure states that $SEM(P^{SEM(P)}) = \emptyset$.

Using our *Reduction*, it is immediate that this principle is a very special instance of the *Extended Cut* introduced in the first article (Remark 2.7). Let us again look at the program

$$P_{ext_Cut}: \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & \neg a \end{array}$$

for WFS$^+$. Since WFS$^+(P_{ext_Cut}) = \{a,\ \neg b\}$, the *Extended Cut* can be applied to $P_{ext_Cut} \cup \{\neg b\}$ and we get $SEM(P_{ext_Cut} \cup \{\neg b\}) \subseteq SEM(P_{ext_Cut})$. But $SEM(P_{ext_Cut} \cup \{\neg b\}) = \{a \leftarrow b\}$ and this program derives $\neg b$ and $\neg a$, which is a contradiction to $a \in SEM(P_{ext_Cut})$. The reason why the *extended Cut* does not hold for WFS$^+$ is that by adding $\neg b$ (as we defined it, namely by just cancelling all rules with heads equal to b), the dependency between a and $\neg a$ gets lost. And it is this dependency that allows WFS$^+$ to derive a. This means that deriving $\neg b$ depends on *first* deriving a. By adding $\neg b$ we did not take into consideration that its derivation depends on others. Our closure condition tells us that we can nevertheless reduce with the whole semantics.

There is another special case of the *Extended Cut* that we consider important:

Definition 5.9 (Weak Model-Property)
If $a \leftarrow \neg b \in P$, *and* $\neg b \in SEM(P)$, *then* $a \in SEM(P)$.

This property ensures with *Reduction*, *PPE* and *Cut* that any semantics SEM(P) is a model of P (the C_1*-property*, see Lemma 5.15).

We will see in Section 5.5 that the *extended Cut* is implied by the *Cut* if *GPPE* holds. Therefore the *Closure* and the *weak Model-Property* become obsolete in the presence of *Cut* and *GPPE*.

5.4 M_P-extension and Transformation

A fundamental condition that one should require is the following: *equivalent* programs should have the same semantics. But this notion of equivalence depends on the underlying semantics: we therefore have to model a very weak form of equivalence, true for all possible semantics. Consider two programs P and Q. If Q is just a renaming of the constants in P, then P and Q should obviously be equivalent, i.e. the renaming of SEM(Q) should equal SEM(P):

Definition 5.10 (Isomorphy)
Let P and Q be isomorphic, i.e. there is an isomorphism I from $B_{\mathcal{L}_P}$ onto $B_{\mathcal{L}_Q}$ s.t. $I(P) = Q$. The principle of Isomorphy states that $I(SEM(P)) = SEM(Q)$.

Together with the properties *Relevance* and PPE *Isomorphy* also applies to more complex programs that need not to be isomorphic but consist of isomorphic subprograms and thus can also be proved to be equivalent.

But there are still other *intuitively correct* equivalences possible between different programs. Let P be a program. If we add $p \leftarrow p$ we certainly get an equivalent program (independently of the truth-ness, falsity or the undefined-ness of p): note that we are modelling the NMR-view (see Example 3.2). Analogously, clauses of the form "$p \leftarrow p, body$" should be useless. For finite propositional programs, all these problems can be solved (in conjunction with some other principles to be introduced later), as far as the Theorems 5.25 and 5.28 are concerned, with the following notion of a *normalform*.

Definition 5.11 (Normalform)
Let P be a program. We define the following transformations:

- T_1*: If $head \leftarrow body$ and $head \leftarrow body'$ are two clauses from P with $body \subseteq body'$, then the clause $head \leftarrow body'$ may be removed.*
- T_2*: Clauses of the form $x \leftarrow x, a_1, \ldots, a_n$, where a_i are atoms, may be removed.*[10]

In this way with any program P a normalform $normalform(P)$ *is associated that is unique up to the order of the clauses and the literals in the clauses. The underlying language of $normalform(P)$ is $\mathcal{L}_P$ and not $\mathcal{L}_{normalform(P)}$*[11].

and the principle of *Equivalence*:

Definition 5.12 (Equivalence)
The principle of Equivalence states that SEM$(P) =$ SEM$(normalform(P))$.

[10] We can as well allow not only atoms a_i but arbitrary literals l_i. This would make our principle stronger. But it turns out that this strengthening is not needed.

[11] We used the same modification in the definition of the reduction with a set of literals (Definition 3.8): if a clause $x \leftarrow x$ is removed and x does not appear in the remaining program, we have to make sure that x is assigned "**f**".

For finite definite propositional programs, we can use *Equivalence* and PPE to construct the model M_P. However, since our normalform-rules are finite rules they can not be applied to e.g. the program consisting of the fact $P(c)$ and the rule

$$P(x) \leftarrow P(s(x)).$$

For infinite propositional programs we therefore still need the explicit requirement that for definite programs any semantics should coincide with the least Herbrand model:

Definition 5.13 (M_P-extension)
If P is a definite program, then SEM(P)=$\{l : l$ a literal with $M_P \models l\}$.

Our next principle is motivated by the program

$$\begin{array}{lll} a & \leftarrow & b, \neg a \\ b & \leftarrow & a, \neg b \end{array}$$

where we want obviously that both $\neg a$ and $\neg b$ are derivable in order to get at least the well-founded semantics. The following principle models the idea that negative literals true in the positive transform P^+ as well as positive literals true in the negative transform P^- of a program should be derivable:

Definition 5.14 (Transformation)
Let P be a program and P^+, P^- as defined in Lemma 3.9. A semantics SEM satisfies the principle of Transformation if the following holds:

- $\neg x \in SEM(P^+) \implies SEM(P \cup \{\neg x\}) \subseteq SEM(P)$,
- $x \in SEM(P^-) \implies x \in SEM(P)$.

Note that in the first condition we not only require that $\neg x \in$ SEM(P) but that a special instance of the *extended Cut* can be applied. We will show in Section 5.5 that *Transformation* follows for finite programs from *GPPE*, *Reduction* and *Relevance*.

Together with the principles introduced in the previous sections, *Transformation* will be powerful enough to simulate WFS.

Figure 4 summarizes the shortcomings of the semantics introduced so far: their failure of some of our weak properties.

5.5 Deducible Rules

We first show that our conditions ensure a semantics to be a model of the underlying program:

Lemma 5.15 (C_1 is derivable, i.e. $SEM(P) \models P$)
A semantics satisfying Reduction*,* weak PPE*,* Cut *and the* Weak Model-Property *also satisfies the following:*

$$\textit{If } a \leftarrow l_1, \ldots, l_n \in P \textit{ and } l_1, \ldots, l_n \in SEM(P) \textit{ then } a \in SEM(P).$$

Proof: Using *weak PPE* and the *Weak Model-Property* we can assume that all l_i are positive. But then we apply *Reduction* and get SEM$(P \cup \{l_1, \ldots, l_n\})$ = SEM$(P^{\{l_1,\ldots,l_n\}}) \cup \{l_1, \ldots, l_n\}$, i.e. (applying Reduction to the fact $a \leftarrow$ again) $a \subseteq$ SEM$(P \cup \{l_1, \ldots, l_n\})$. Using *Cut*, the last set is contained in SEM(P) and we are done. ∎

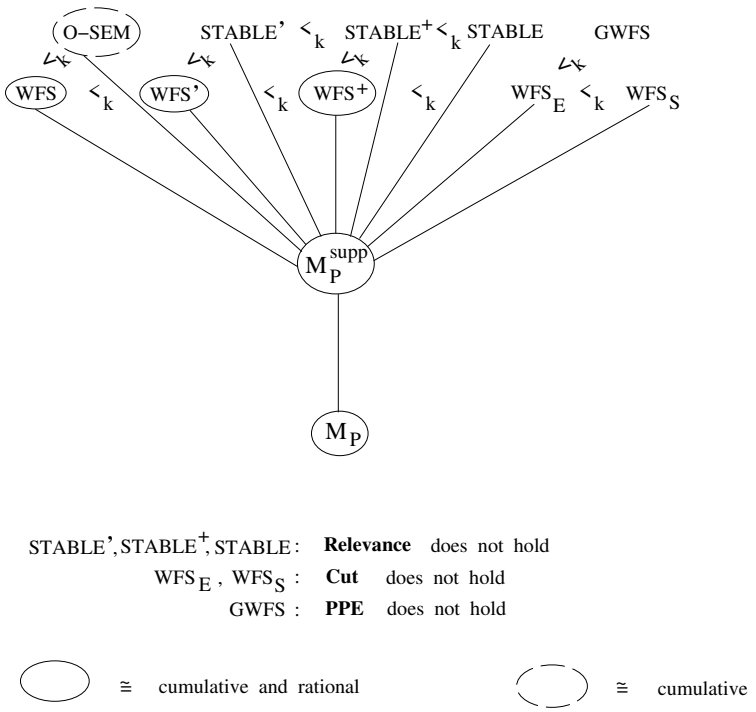


Figure 4: Semantics wrt Relevance, Cut and PPE

Lemma 5.16 (Deducible variant of Modularity)
Let P be instantiated, $P = P_1 \cup P_2$, $B_{P_1} \cap B_{P_2} = \{A\}$ *and* $P_2 = rel_rul(P, A)$.
Using Modularity, Relevance *and* Reduction *we have:*

- *if* $A \in$ *SEM*(P_2) *then* *SEM*$(P) =$ *SEM*$(P_1 \cup \{A\}) \cup$ *SEM*(P_2),
- *if* $\neg A \in$ *SEM*(P_2) *then* *SEM*$(P) =$ *SEM*$(P_1 \cup \{\neg A\}) \cup$ *SEM*(P_2).

If SEM(P_2) is *complete*, i.e. $\forall x \in B_{P_2} : \ x \in$ SEM(P_2) or $\neg x \in$ SEM(P_2), then we have:

Lemma 5.17 (Further variant of Modularity)
Let P be instantiated, $P = P_1 \cup P_2$, $B_{P_1} \cap B_{P_2} = \{A\}$, $P_2 = rel_rul(P, A)$ *and* *SEM*(P_2) *complete.*
Using Modularity, Relevance *and* Reduction *we have:*

$$\textit{SEM}(P) = \textit{SEM}({P_1}^{SEM(P_2)}) \cup \textit{SEM}(P_2)$$

Modularity is a restricted form of *Cumulativity* if we assume some of our principles introduced above:

Lemma 5.18 (Extended Cumulativity implies Modularity)
From Reduction, Relevance *and* Extended Cumulativity *follows Modularity.*

Proof: Let the assumptions of *Modularity* be satisfied. We apply *Relevance*, *Cumulativity*, *Reduction* and get

$$\text{SEM}(P) = \text{SEM}(P \cup \text{SEM}(P_2)) = \text{SEM}(P^{SEM(P_2)}) \cup \text{SEM}(P_2).$$

Since our definition of *Reduction* P^M is idempotent ($(P^M)^M = P^M$), we have:

$$\text{SEM}(P^{SEM(P_2)}) \cup \text{SEM}(P_2) = \text{SEM}(\ P_1^{SEM(P_2)^{SEM(P_2)}} \cup P_2^{SEM(P_2)}\ )\ \cup\ \text{SEM}(P_2),$$

which equals $\text{SEM}(\ (P_1^{SEM(P_2)} \cup P_2)^{SEM(P_2)})\ \cup\ \text{SEM}(P_2)$ by the definition of P^M. Finally we have

$$\begin{aligned} \text{SEM}(\ (P_1^{SEM(P_2)} \cup P_2)^{SEM(P_2)})\ \cup\ \text{SEM}(P_2) &= \text{SEM}(P_1^{SEM(P_2)} \cup P_2 \cup SEM(P_2)) \\ &= \text{SEM}(P_1^{SEM(P_2)} \cup P_2). \end{aligned}$$

The first identity is obtained by applying *Reduction* ($M := \text{SEM}(P_2)$, $P := P_1^{SEM(P_2)} \cup P_2$). The second identity follows by applying *Relevance*: the elements of B_{P_2} only depend on P_2 (note that we are proving *Modularity* and this is our general assumption) and by applying *Cumulativity* to P_2. ■

Obviously, our versions of partial evaluation imply each other:

Lemma 5.19 (Weak PPE is deducible)
If Relevance *holds then* GPPE *implies* PPE *and* PPE *implies* weak PPE.

Hu and Yuan defined in [HY91] the principle of locality which seems to be very strongly related (maybe even equivalent) to the conjunction of our *Relevance* and *Modularity*. They also considered in this paper programs with only negative literals in their bodies. For finite propositional programs GPPE together with *Equivalence* ensures us such a normal form:

Lemma 5.20 (GPPE implies a strong normalform)
Let SEM satisfy Equivalence *and* GPPE*. Then any finite propositional program P can be equivalently transformed into a program P^* where only negative literals occur in the bodies of rules. P^* is called the strong normalform of P.*

Proof: The proof is easy. Simply replace iteratively any positive occurrence of an atom a in a rule by applying GPPE (before applying GPPE, make sure that *Equivalence* has been applied to avoid introducing another positive occurrence of a). ■

Lemma 5.21 (GPPE implies extended versions of Cut and Cumulativity)
Let SEM satisfy Reduction*,* Equivalence*,* Relevance*,* weak Model-Property *and* GPPE*.*

If SEM satisfies Cut *then it also satisfies* Extended Cut *for all programs in strong normalform.*

If SEM satisfies Cumulativity *then it also satisfies* Extended Cumulativity *for all programs in strong normalform.*

Proof: Let $\neg x \in \text{SEM}(P)$. Then by *Reduction*

$$\text{SEM}(P \cup \{\neg x\}) = \text{SEM}(P^{\{\neg x\}}) \cup \{\neg x\}.$$

Let $P[\frac{\mathbf{t}}{\neg x}]$ be the program obtained from P by replacing any occurrence of $\neg x$ with true. Using *Relevance* we have (here it is neccessary to assume that P is in strong normalform):

$$\text{SEM}(P^{\{\neg x\}}) = \text{SEM}(P[\frac{\mathbf{t}}{\neg x}]),$$

as far as literals different from $\{x, \neg x\}$ are concerned (we assume this for the rest of this proof to simplify notation). Since also $P[\frac{\mathbf{t}}{\neg x}] = (P[\frac{a}{\neg x}] \cup \{a \leftarrow \neg x\})^{\{a\}}$ (where a is a new constant-symbol) we get by using *weak PPE*

$$\text{SEM}(P \cup \{\neg x\}) = \text{SEM}((P[\frac{a}{\neg x}] \cup \{a \leftarrow \neg x\})^{\{a\}}).$$

Adding $\{a\}$ on the right-hand side and then applying *Reduction* we get

$$\subseteq \text{SEM}((P[\frac{a}{\neg x}] \cup \{a \leftarrow \neg x\})^{\{a\}}) \cup \{a\} = \text{SEM}(P[\frac{a}{\neg x}] \cup \{a \leftarrow \neg x\} \cup \{a\}).$$

We can now apply the nonextended Cut (note that $a \in \text{SEM}(P[\frac{a}{\neg x}] \cup \{a \leftarrow \neg x\})$ because of the weak Model-Property) and we get

$$\subseteq \text{SEM}(P[\frac{a}{\neg x}] \cup \{a \leftarrow \neg x\}) = \text{SEM}(P) \cup \{a\}.$$

Therefore, $\text{SEM}(P \cup \{\neg x\}) \subseteq \text{SEM}(P)$.

A similar reasoning proves the second part of the Lemma.

■

Lemma 5.22 (Simulating Transformation for finite programs)
For finite programs GPPE*,* Reduction*, and* Equivalence *imply* Transformation *for programs in strong normalform.*

Proof: For programs in strong normalform, P^- consists of exactly the facts of P, therefore the second condition is trivially satisfied by *Reduction*.

If $\neg x \in \text{SEM}(P^+)$ then x does not occur in any head of P. This means that $\text{SEM}(P \cup \{\neg x\}) = \text{SEM}(P)$.

■

Finally, also M_P-extension follows for finite programs from *PPE*, *Reduction*, and *Equivalence*:

Lemma 5.23 (Simulating M_P-extension for finite propositional programs)
For finite programs, M_P-extension *follows from* weak PPE*,* Reduction *and* Equivalence*.*

Proof: Let P be a finite, definite propositional program. We iterate the following steps to obtain a series of programs and eventually arrive at the empty program.

The first step consists in reducing the program by evaluating the clauses with empty bodies (using *Reduction*). We can also evaluate all atoms that do not occur in any heads to **f** (again by *Reduction*).

The second step consists in applying *Weak PPE* to reduce the program further and generating clauses of the form

$$x \leftarrow x, \ldots.$$

These clauses can be cancelled using *Equivalence*. We then restart with the first step amd apply it to the current program.

■

5.6 Main definition and results

We are now in a position to define what a reasonable semantics is:

Definition 5.24 (Well-behaved semantics)
A well-behaved semantics SEM is a mapping

$$\{P : P \text{ a program}\} \longrightarrow 2^{Lit}$$

such that the following conditions are satisfied: Cut, Closure, weak Model-Property, Isomorphy, M_P-extension, Transformation, Relevance, Reduction, PPE *and* Modularity.

If we only consider finite propositional programs and we also assume *GPPE* then we are left with only: *Reduction*, *Equivalence*, *Cut*, *Relevance*, *Isomorphy*, *weak Model-Property* and *Modularity*. In addition, it is sufficient to state the last five conditions only for programs in strong normalform.

Further work is needed to derive other properties that are implied by our conditions. Our idea is that fixing a semantics on a *reasonable* class of programs (for example axiomatically requiring SEM($P_{splitting\ I}$)=$\emptyset$, SEM($\{a \leftarrow \neg b, b \leftarrow \neg c, c \leftarrow \neg a\}$)=$\emptyset$ and further axioms) already determines these semantics using our conditions. It is, however, possible that we still need more conditions (for different axiomatisations).

Using the principles we have introduced so far, we are able to prove:

Theorem 5.25 (Well-behaved semantics are extensions of M_P^{supp})
Any well-behaved semantics SEM coincides for stratified programs with M_P^{supp}.

Proof: We use the following two steps (in the indicated order) to reduce a program P (and determining the truthvalue of any atom of $\mathcal{L}_P$) until the empty program is obtained:

1. Iterating *Reduction* we can assume that P (after *Reduction*) has the following two properties
   - no clause of P has an empty body, and
   - all atoms occurring in P already occur in some head.
2. Let M^{pos} be the set of all atoms that do not depend negatively on another atom. We can split P into
   $$P = P_1 \cup P_2,$$
   where P_2 consists of all clauses that are solely built from atoms contained in M^{pos}.

We now get (using M_P*-extension*)

$$\text{SEM}(P_2) = \{\neg x : \ x \in M^{pos}\}$$

which is in accordance with M_P^{supp}: note that P can be stratified in such a way that P_2 is just in the first stratum. By *Modularity* P is reduced to the smaller

$$P' = P_1^{SEM(P_2)}.$$

We can iterate our procedure (first using 1. then 2.) and, because P is stratified (no atom depends negatively on itself) the procedure stops after a finite number of steps (at most after

n steps where n is the number of strata) with the empty program. All atoms from P are evaluated according to M_P^{supp}.

Note that we only used *Reduction*, *Modularity* and M_P*-extension* in our proof.

For finite propositional programs M_P*-extension* can be replaced by *PPE* and *Equivalence* (note that *Reduction* is already used).

■

Using Lemma 5.22 and 5.23 we have the following

Corollary 5.26 (The Case of finite propositional programs)
For finite propositional *programs the last theorem also holds for semantics satisfying only* Reduction*,* Modularity*,* weak PPE *and* Equivalence*.*

Corollary 5.27
Any well-behaved semantics SEM coincides for locally-stratified programs with Przymusinski's unique perfect model.

The proof of the corollary parallels the proof of the theorem and is therefore omitted.

Theorem 5.28 (WFS is the weakest well-behaved extension of M_P^{supp})
Any well-behaved semantics is a $\leq_k$-extension of WFS.

Proof: Let SEM be well-behaved. We use some of our principles to simulate the construction of the well-founded semantics in Lemma 3.9. In the following we only consider the case of successor ordinals $\alpha + 1$ since the case of limit ordinals follows easily. We first note that

1. $M^+_{\alpha+1} \subseteq \text{SEM}(P_\alpha)$. This is clear because $M^+_{\alpha+1} \subseteq \text{SEM}(P_\alpha)^+$ (by M_P*-extension* and construction of $M^+_{\alpha+1}$) and *Transformation*:

$$M^+_{\alpha+1} \subseteq \text{SEM}(P_\alpha \cup M^+_\alpha) \subseteq \text{SEM}(P_\alpha).$$

2. $M^-_{\alpha+1} \subseteq \text{SEM}(P_\alpha)$. This is obvious because $M^-_{\alpha+1} \subseteq \text{SEM}(P_\alpha)^-$ (by M_P*-extension* and construction of $M^-_{\alpha+1}$) and *Transformation*:

$$M^-_{\alpha+1} \subseteq \text{SEM}(P_\alpha).$$

Using *Reduction* we get

$$\text{SEM}(P_\alpha^{M^+_{\alpha+1} \cup M^-_{\alpha+1}}) \cup M^+_{\alpha+1} \cup M^-_{\alpha+1} = \text{SEM}(P_\alpha \cup M^+_{\alpha+1} \cup M^-_{\alpha+1}).$$

Because of **2.** we have

$$M^-_{\alpha+1} \subseteq \text{SEM}(P^+_\alpha = \text{SEM}(P^+_\alpha \cup M^-_\alpha = \text{SEM}((P_\alpha \cup M^-_\alpha)^+$$

and therefore (with *Transformation*)

$$\text{SEM}((P_\alpha \cup M^-_\alpha) \cup M^+_\alpha) \subseteq \text{SEM}(P_\alpha \cup M^-_\alpha) \subseteq \text{SEM}(P_\alpha).$$

The last subset-relation follows by applying *Cut*.

Therefore we have proven that

$$\text{SEM}(P_{\alpha+1}) \subseteq \text{SEM}(P_\alpha),$$

i.e. $\text{WFS}(P) \leq_k \text{SEM}(P)$ in view of Lemma 3.9.

Note that we do not need properties of *Isomorphy*, *PPE*, *Modularity* and *Equivalence* in the proof.

■

Using Lemma 5.22 and 5.23 we have the following

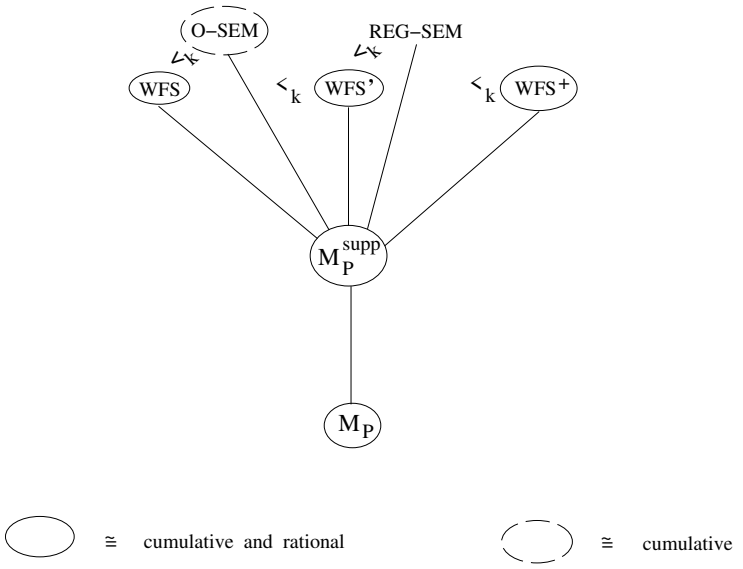


Figure 5: Well-behaved Semantics for Normal Programs

Corollary 5.29 (The Case of finite propositional programs)
For finite propositional *programs the last theorem also holds for semantics satisfying only* Reduction, GPPE, Equivalence *and* Cut. Relevance *is not needed and the theorem therefore also applies to STABLE.*

We note that most of our semantics are well-behaved. The proofs are sometimes tedious and technical and will therefore be omitted.

Lemma 5.30 (Well-behavedness of some semantics)
The semantics WFS, WFS', WFS$^+$, O-SEM and REG-SEM are well-behaved.

In Example 3.6 we showed that the stable semantics is not cumulative (and that this does not depend on the nonexistence of stable models). The program $P_{Makinson}$ defined there is also a good example to show the failure of Cumulativity for STABLE^{rel}, a version of STABLE (defined in [DM94b]) that satisfies all of our reasonable conditions. Thus our *strong* property of Cumulativity does not follow from our *weak* reasonable properties (REG-SEM is an additional candidate):

Lemma 5.31 (Well-behaved does not imply Cumulativity)
STABLE^{rel} is a well-behaved semantics that is not cumulative.

Similarly, Example 4.5 shows that well-behaved semantics having the GCWH property cannot satisfy the *Extended Cut* or even C_3:

Lemma 5.32 (Well-behaved and GCWH-property)
Let SEM be well-behaved.
a) *GCWH is incompatible with* Extended Cut.
b) *GCWH is incompatible with* C_3.

5.7 Representation conjectures

We claim that our reasonable conditions together with the property of *Rationality* exactly describe the semantics WFS, WFS$^+$ (=WFS$_C$) and WFS$'$. In particular, we have the following conjectures:

Conjecture 5.33 (Characterization of WFS, WFS$^+$ and WFS$'$)
There are no well-behaved and rational semantics other than WFS, WFS$^+$ and WFS$'$.

Conjecture 5.34 (WFS$^+$)
WFS$^+$ is the only well-behaved and rational semantics satisfying Supraclassicality.

Conjecture 5.35 (Characterization of WFS and WFS$'$)
There are no well-behaved and rational semantics satisfying C_3 other than WFS and WFS$'$.

Figure 5 summarizes and illustrates the relationship between well-behaved semantics.

6 Concluding remarks

Our starting point in this article was the observation that certain semantics recently defined showed some strange behaviour. This led us to formulate a list of *weak* principles where, as we argue, all semantics should be checked against (*PPE*, *Modularity*, *Relevance*, *Cut* etc.). We have therefore presented, to the best of our knowledge for the first time, an axiomatic framework for reasoning about and defining semantics of logic programs. So it is possible to ask for all instances satisfying certain properties (eg. the *strong* principles introduced in [Dix94]) and to state (and eventually prove) interesting representation theorems of the kind stated in Section 5.7.

We also strongly believe that our principles can be used for an efficient implementation of these semantics. First results were already obtained (also for disjunctive programs) in [DM92b, MD93] as well as recently in [BD94d, BD94a, BD94b] and will be investigated in a forthcoming papers. As to disjunctive programs, the reader is referred to the (forthcoming) third article in this series or to [Dix92b, DM94a].

Acknowledgements

Discussions with Heinrich Herre and useful suggestions of John Schlipf helped to improve this paper a lot. Many thanks to Martin Müller who proofread the paper and made a lot of helpful remarks during many talks.

The author had some very interesting discussions with Li-Yan Yuan who made some important observations. This led to a careful reformulation of the original Transformation-condition which turned out to be too strong. Jia-Huai You and Li-Yan Yuan are currently working on an extension of the well-behaved principles to extended normal logic programs. They suggest an interesting set of properties which might be equivalent to our well-behaved conditions plus the GPPE.

Last but not least an anonymous referee read the paper extremely carefully and still found lots of inaccuracies which I hope to have cleaned up now.

References

[AB94] K.R. Apt and R. Bol. Logic Programming and Negation: A Survey. *Journal of Logic Programming*, 19-20:9–71, 1994.

[ABW88] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases*, chapter 2, pages 89–148. Morgan Kaufmann, 1988.

[BD94a] Stefan Brass and Jürgen Dix. A Caracterization of the Stable Semantics by Partial Evaluation. Technical Report TR 15/94, University of Koblenz, Department of Computer Science, Rheinau 1, September 1994. submitted.

[BD94b] Stefan Brass and Jürgen Dix. A General Approach to Bottom-Up Computation of Disjunctive Semantics. Technical Report TR 11/94, University of Koblenz, Department of Computer Science, Rheinau 1, September 1994. submitted.

[BD94c] Stefan Brass and Jürgen Dix. A disjunctive semantics based on unfolding and bottom-up evaluation. In Bernd Wolfinger, editor, *Innovationen bei Rechen- und Kommunikationssystemen, (IFIP-Congress, Workshop FG2: Disjunctive Logic Programming and Disjunctive Databases)*, pages 83–91. Springer-Verlag, 1994.

[BD94d] Stefan Brass and Jürgen Dix. Disjunctive Semantics based upon Partial and Bottom-Up Evaluation. Technical Report TR 16/94, University of Koblenz, Department of Computer Science, Rheinau 1, September 1994. submitted.

[BF91a] Nicole Bidoit and Christine Froidevaux. General logical Databases and Programs: Default Logic Semantics and Stratification. *Information and Computation*, 91:15–54, 1991.

[BF91b] Nicole Bidoit and Christine Froidevaux. Negation by Default and unstratifiable logic Programs. *Theoretical Computer Science*, 78:85–112, 1991.

[BLM90] Chitta Baral, Jorge Lobo, and Jack Minker. Generalized Well-founded Semantics for Logic Programs. In M. E. Stickel, editor, *10th International Conference on Automated Deduction, LNAI 449, subseries LNCS*, pages 102–116. Springer, J. Siekmann, July 1990.

[CK91] Jianhua Chen and Sukhamay Kundu. The Strong Semantics For Logic Programs. In Z.W. Ras and M. Zemankova, editors, *Proceedings of the 6th Int. Symp. on Methodologies for Intelligent Systems, Charlotte, NC, 1991*, pages 490–499. Springer, Lecture Notes in Artificial Intelligence 542, 1991.

[Dix91a] Jürgen Dix. Classifying Semantics of Logic Programs. In Anil Nerode, Wiktor Marek, and V. S. Subrahmanian, editors, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, pages 166–180. Washington D.C, MIT Press, July 1991.

[Dix91b] Jürgen Dix. Cumulativity and Rationality in Semantics of Normal Logic Programs. In J. Dix, K. P. Jantke, and P. H. Schmitt, editors, *Proceedings of the first Workshop on Nonmonotonic and Inductive Logic 1990 in Karlsruhe*, pages 13–37. LNCS 543, Springer, October 1991.

[Dix92a] Jürgen Dix. A Framework for Representing and Characterizing Semantics of Logic Programs. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, pages 591–602. San Mateo, CA, Morgan Kaufmann, 1992.

[Dix92b] Jürgen Dix. Classifying Semantics of Disjunctive Logic Programs. In K. Apt, editor, *LOGIC PROGRAMMING: Proceedings of the 1992 Joint International Conference and Symposium*, pages 798–812. MIT Press, November 1992.

[Dix92c] Jürgen Dix. *Nichtmonotones Schließen und dessen Anwendung auf Semantiken logischer Programme*. PhD thesis, University of Karlsruhe, 1992.

[Dix94] Jürgen Dix. A Classification-Theory of Semantics of Normal Logic Programs: I. Strong Properties. *Fundamenta Informaticae*, forthcoming, 1994.

[Dix95] Jürgen Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In Andre Fuhrmann and Hans Rott, editors, *Logic, Action and Information. Proceedings of the Konstanz Colloquium in Logic and Information (LogIn '92)*, pages 227–313. DeGruyter, 1995.

[DK94] Jürgen Dix and Martin Kummer. Herbrand's Theorem for Nonstandard Inference Operations. *Journal of Logic and Computation*, Vol. 4, No. 2:143–156, 1994.

[DM92a] Jürgen Dix and David Makinson. The Relationship between KLM and MAK models for Nonmonotonic Inference Operations. *Journal of Logic, Language and Information*, Volume 1, No. 2:131–140, 1992.

[DM92b] Jürgen Dix and Martin Müller. Abstract Properties and Computational Complexity of Semantics for Disjunctive Logic Programs. In *Proc. of the Workshop W1, Structural Complexity and Recursion-theoretic Methods in Logic Programming, following the JICSLP '92*, pages 15–28. H. Blair and W. Marek and A. Nerode and J. Remmel, November 1992. also available as Technical Report 13/93, University of Koblenz, Department of Computer Science.

[DM93] Jürgen Dix and Martin Müller. Implementing Semantics for Disjunctive Logic Programs Using Fringes and Abstract Properties. In Luis Moniz Pereira and Anil Nerode, editors, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Second International Workshop*, pages 43–59. Lisbon, MIT Press, July 1993.

[DM94a] Jürgen Dix and Martin Müller. An Axiomatic Framework for Representing and Characterizing Semantics of Disjunctive Logic Programs. In Pascal Van Hentenryck, editor, *Proceedings of the 11th Int. Conf. on Logic Programming, S. Margherita Ligure*, pages 303–322. MIT, June 1994.

[DM94b] Jürgen Dix and Martin Müller. Partial Evaluation and Relevance for Approximations of the Stable Semantics. In Z.W. Ras and M. Zemankova, editors, *Proceedings of the 8th Int. Symp. on Methodologies for Intelligent Systems, Charlotte, NC, 1994*, pages 511–520. Springer, Lecture Notes in Artificial Intelligence 869, 1994.

[DM94c] Jürgen Dix and Martin Müller. The Stable Semantics and its Variants: A Comparison of Recent Approaches. In L. Dreschler-Fischer and B. Nebel, editors, *Proceedings of the 18th German Annual Conference on Artificial Intelligence (KI '94), Saarbrücken, Germany*, pages 82–93. Springer, Lecture Notes in Artificial Intelligence 861, 1994.

[Dun91] P. M. Dung. Negation as Hypotheses: An Abductive Foundation for Logic Programming. In Koichi Furukawa, editor, *Proceedings of the 8th Int. Conf. on Logic Programming, Paris*. MIT, June 1991.

[GL88] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[HY91] Yong Hu and Li Yan Yuan. Extended Well-Founded Model Semantics for General Logic Programs. In Koichi Furukawa, editor, *Proceedings of the 8th Int. Conf. on Logic Programming, Paris*, pages 412–425. MIT, June 1991.

[KLM90] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence*, 44(1):167–207, 1990.

[Mak89] David Makinson. General Theory of Cumulative Inference. In Reinfrank, de Kleer, Ginsberg, and Sandewall, editors, *Non-Monotonic Reasoning*. Lecture Notes in Artificial Intelligence 346, Springer-Verlag, January 1989.

[Mak94] David Makinson. General Patterns in Nonmonotonic Reasoning. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming Vol. 2, Nonmonotonic and Uncertain Reasoning*, chapter 2.2. Oxford University Press, 1994.

[MD93] Martin Müller and Jürgen Dix. Implementing Semantics for Disjunctive Logic Programs Using Fringes and Abstract Properties. In Luis Moniz Pereira and Anil Nerode, editors, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Second International Workshop*, pages 43–59. Lisbon, MIT Press, July 1993.

[Min93] Jack Minker. An Overview of Nonmonotonic Reasoning and Logic Programming. *Journal of Logic Programming, Special Issue*, 17, 1993.

[PAA92] L.M. Pereira, J.J. Alferes, and J.N. Aparicio. Adding Closed World Assumptions to Well Founded Semantics. In ICOT, editor, *Fifth Generation Computer Systems, Proceedings of the Conference*, pages 562–569. OMSHA, June 1992.

[Prz88] Teodor Przymusinski. On the declarative semantics of deductive databases and logic programs. In Jack Minker, editor, *Foundations of Deductive Databases*, chapter 5, pages 193–216. Morgan Kaufmann, 1988.

[Prz89] Teodor Przymusinski. Every Logic Program has a natural Stratification and an iterated fixed point Model. In *Proc. of the Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania*, pages 11–21. ACM Press, March 1989.

[Prz90] Teodor Przymusinski. Well-founded Semantics Coincides With Three-Valued Stable Semantics. *Fundamenta Informaticae*, XIII:445–463, 1990.

[Prz91] T. C. Przymusinski. Three-valued non-monotonic formalisms and semantics of logic programs. *Artificial Intelligence*, 49:309–343, 1991. (Extended abstract appeared in: Three-valued non-monotonic formalisms and logic programming. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89), Toronto, Canada*, pages 341–348, Morgan Kaufmann, 1989.).

[Prz92] T. C. Przymusinski. Two simple characterizations of well-founded semantics. In I. Havel and V. Koubek, editors, *Proceedings of the International Conference on Mathematical Foundations of Computer Science (MFCS'92)*, pages 451–462, Prague, Czechoslovakia, 1992. Springer.

[Sch92] John S. Schlipf. Formalizing a Logic for Logic Programming. *Annals of Mathematics and Artificial Intelligence*, 5:279–302, 1992.

[SS94] Chiaki Sakama and Hirohisa Seki. Partial Deduction of Disjunctive Logic Programs: A Declarative Approach. In *Fourth International Workshop on Logic Program Synthesis and Transformation (LOPSTR'94)*. Lecture Notes in Computer Science, Springer-Verlag, July 1994.

[SZ91] D. Sacca and C. Zaniolo. Partial models and Three-Valued Models in Logic Programs with Negation. In Anil Nerode, Wiktor Marek, and V. S. Subrahmanian, editors, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, pages 87–104. Washington D.C, MIT Press, July 1991.

[vGRS88] Allen van Gelder, Kenneth Ross, and J.S. Schlipf. Unfounded Sets and well-founded Semantics for general logic Programs. In *Proceedings 7th Symposion on Principles of Database Systems*, pages 221–230, 1988.

[vGRS91] Allen van Gelder, Kenneth Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.

[YY90] Jia-Huai You and Li-Yan Yuan. Three-valued Formalization of Logic Programming: is it needed. In *Proc. of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,*, pages 172–182. ACM Press, 1990.

[YY93] Jia-Huai You and Li-Yan Yuan. On the Equivalence of Semantics for Normal Logic Programs. In *Proceedings of Workshop on Logic Programming with Incomplete Information, Vancouver Oct. 1993, following ILPS' 93*, pages 161–176, 1993. available as Research Report of ISIS-RR-93-E12, Fujitsu Labs.

Contents

1 Introduction **1**
- 1.1 Our approach . . . 1
- 1.2 Organization of the paper . . . 3

2 Notation and terminology **3**
- 2.1 Logic programs and three-valued logic . . . 3
- 2.2 Sceptical semantics $\text{SEM}_P^{scept}(U)$ . . . 4

3 The classical semantics **5**
- 3.1 The supported semantics M_P^{supp} . . . 6
- 3.2 STABLE . . . 7
- 3.3 WFS . . . 8

4 Extensions of WFS **10**
- 4.1 GWFS, EWFS . . . 11
- 4.2 WFS_C, WFS_E, WFS_S . . . 12
- 4.3 The relationship between the semantics . . . 14

5 Well-behaved semantics **15**
- 5.1 Relevance and Reduction . . . 16
- 5.2 PPE and Modularity . . . 17
- 5.3 Closure Conditions . . . 19
- 5.4 M_P-extension and Transformation . . . 20
- 5.5 Deducible Rules . . . 21
- 5.6 Main definition and results . . . 25
- 5.7 Representation conjectures . . . 28

6 Concluding remarks **28**

Bibliography **28**