The Stable Semantics and its Variants: A Comparison of Recent Approaches

Jürgen Dix[1] and Martin Müller[2]

[1] University of Koblenz, Dept. of Computer Science, Rheinau 1, D-56075 Koblenz, Germany dix@informatik.uni-koblenz.de
[2] Graduiertenkolleg Kognition and German Research Center for AI (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken mmueller@dfki.uni-sb.de

Abstract. The stable and the well-founded semantics are among the leading semantics for logic programs. While stable models do not always exist (and therefore no meaningful inferences can be drawn), the (three-valued) well-founded model is always defined. A weakness of WFS on the other hand is its inability to allow for *reasoning-by-cases*.
We claim that - besides the inconsistency problem - the reason of the anomalous behavior of STABLE is the failure of *Relevance* – a principle introduced in previous work of the first author.
We define, discuss and compare various extensions of STABLE and WFS under both aspects. In particular, we determine the relationship with the approach recently introduced by J. Schlipf. We use our own approach to show that Schlipf's semantics WFS_C coincides with Dix' WFS^+.

1 Introduction and Motivation

The well-founded semantics and the stable semantics are undoubtedly the most influential semantics for normal logic programs. One may almost speak about two schools of how to give meaning to programs with negation, observing the frequent and vehement discussions between followers of both semantics, be it users or developers. Our impression is that both sides often miss to clarify their general intent in using a particular semantics, which is the reason for a number of misunderstandings.

A semantics like the stable model semantics STABLE [GL88], associating a set of classical models with a given program, may serve three purposes: It may

1. answer the question of what is necessarily true (true in *all* models), or
2. answer the question of what is possibly true (true in *at least one* model), or
3. offer every model as *one complete scenario* satisfying the constraints imposed by the program.

The first two are usually called the *sceptical* and *credulous* view of a semantics. The third may be called *nondeterministic* as by Torres [Tor93], who makes the same distinction. We prefer to talk, more intuitively, about *scenario* semantics.

The well-founded semantics WFS [vGRS91] associates a single three-valued model with any normal program. Therefore WFS is inherently sceptical and all three views sketched above coincide. This means that a formal comparison between STABLE and WFS can only be performed using the sceptical semantics induced by STABLE. The program begun by Dix [Dix91, Dix92] follows this path.

That STABLE *may* be used as a scenario semantics is often pointed out as a major advantage of STABLE over WFS (see e.g. [Tor93]), and illustrated by the following example:

Example 1 Cafeteria-Lounge Scenario.
Imagine a network of hotels buildings *a,b,c,*..., defined by a set of facts of the form *adjacent(a,b)*. The problem is how to supply every building with either a cafeteria or a lounge, but not both, such that: *i)* every building without a cafeteria is adjacent to a building with one, and *ii)* no two adjacent buildings have a cafeteria. This may be captured by the following program, supplemented by the definition of adjacency:

$$\begin{aligned} P_{lounge}: \quad lounge(X) &\leftarrow adjacent(X,Y), cafeteria(Y) \\ cafeteria(X) &\leftarrow \neg lounge(X) \end{aligned}$$

The stable models of this program are exactly the solutions of the problem, namely the possible scenarios under the given constraints.

WFS essentially returns the adjacency facts. This weakness, together with the inability to express case-based reasoning, is the major objection against the well-founded semantics:

Example 2 Case-Splitting.

$$\begin{aligned} P_{splitting}: \; & a \leftarrow \neg b \\ & b \leftarrow \neg a \\ & p \leftarrow a \\ & p \leftarrow b \end{aligned}$$

Although neither a nor b can be "derived" in a semantics based on two-valued models the disjunction $a \vee b$, hence also p, is true. STABLE as well as completion semantics handle this program accordingly. WFS, however, leaves p undefined.

On the other hand WFS is defined on *all* normal programs, while STABLE may turn out to be inconsistent: in Example 1, we just have to add the trivial negative loop $p \leftarrow \neg p$ to P_{lounge}.

STABLE suffers from the *brute force* inconsistency handling known from first order logic: Just a *small inconsistency* in a possibly large set of axioms makes *everything* classically derivable from them, which is as good as nothing. This makes pure first order logic essentially unusable for practical purposes in knowledge representation and logic programming, and we claim that the same holds for the *pure* version of STABLE. This does not affect, however, the success of the stable semantics in establishing links between nonmonotonic reasoning and logic programming [MS89] and its fertilizing effect on abstract investigations into logic program semantics.

An important objection may be that it sometimes *is* intended that a program have no models if certain integrity constraints are not satisfied. In the particular case of STABLE, integrity constraints even belong to the essence of the semantics [FLMS93]. Yet, also integrity constraints are usually relevant only to a part of the program and failure of one constraint should not lead the whole thing astray; for instance, in order to support localization of errors. Furthermore, we feel that integrity constraints add a level of complexity to a semantics and should be understood independently.

In the sequel we define, discuss, and compare a number of extensions of STABLE that avoid the mentioned shortcomings. We also show that the somewhat anomalous behavior of STABLE is due to the failure of *Relevance* which cannot be repaired in an obvious or straightforward manner.

The plan of the paper is as follows: In Section 2 below we give some notation and briefly review the stable and the wellfounded semantics. Section 3 recalls the abstract approach to what we call a *well-behaved* semantics and lays ground for the subsequent sections. Section 4 applies our abstract approach to define elegantly various extensions of STABLE and WFS. We show that the semantics proposed by Schlipf recently ([Sch92]) are instances in our framework. We also discuss the most important variants of STABLE recently proposed and determine their interrelationships. Section 5 addresses related attempts and concludes.

In a companion paper ([DM94]), we employ our approach to define STABLE^{rel}, a version of STABLE satisfying the *Relevance* condition while staying close to STABLE in spirit. There, we also discuss different versions of the *unfolding* property used in logic programming.

2 Notation

A program clause or a *rule* is a formula $a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots \neg c_l$, where $n \geq 1$ and $m, l \geq 0$. As usual, the comma represents conjunction. A *rule* is called *positive* or *definite* rule if $l = 0$. a is said to be the *head* of the rule, $b_1, \ldots, b_m, \neg c_1, \ldots \neg c_l$ its *body*. A *program* P is a finite set of rules and it straightforwardly inherits the typology of clauses. The *Herbrand base* induced by a program P is denoted B_P. In general the a, b_j, c_l are arbitrary atomic formulæ. But to make our point here we join the community which restricts itself to fully grounded programs. If we have no function symbols, i.e., deal only with so-called *datalog* programs, and at least one constant symbol, then the grounded program is finite iff the original program is. In the sequel *program* will mean *fully grounded program* (which may be infinite).

A program P induces a notion of *dependency* between atoms from B_P. We will say *a depends immediately* on b if b appears in the body of a clause in P, such that a appears in its head. The two place relation *depends on* is defined as the transitive closure of *depends immediately on*. We will need the notions *dependencies of* and *rules relevant for an atom* x:

- $\mathit{dependencies\_of}(x) := \{a : x \text{ depends on } a\}$,
- $\mathit{rel\_rul}(P, x)$ is the set of *relevant rules* of P with respect to x, i.e., the set of rules that contain an $a \in \mathit{dependencies\_of}(x)$ in their heads,

These definitions extend straightforwardly to arbitrary literals by setting $\mathit{dependencies\_of}(\neg x) := \mathit{dependencies\_of}(x)$ and $\mathit{rel\_rul}(P, \neg x) := \mathit{rel\_rul}(P, x)$.

Our notion of a semantics is very general ([Dix91]):

Definition 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:

$$\mathrm{SEM}_P \subseteq \mathrm{MOD}_{3-val}^{Herb_{\mathcal{L}_P}}(P).$$

We associate with any semantics SEM a sceptical entailment relation SEM^{scept}:

Definition 2 Sceptical entailment relation $\mid\!\sim_P$.
Let P be a program and U a set of atoms. Any semantics SEM induces a sceptical entailment relation SEM_P^{scept} as follows:

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

Since our main emphasis lies on the sceptical semantics, we will henceforth use SEM instead of SEM^{scept}.

We have to distinguish between two different notions of one semantics being an *extension* of another one:

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

For instance, STABLE extends WFS in the former sense while both STABLE and WFS extend the supported semantics M_P^{supp} ([ABW88]) in the latter sense.

2.1 Stable Semantics vs. Well-founded Semantics

One of the central ideas of the stable semantics is that any atom in a potential model should have a definite reason to be true or false. This idea was made explicit by Bidoit and Froidevaux [BF91] and, independently, by Gelfond and Lifschitz [GL88] as follows: for a program P and a model $N \subseteq B_P$ we define

$$P^N := \{\mathit{rule}^N : \mathit{rule} \in P\}, \qquad \text{where}$$

$$(a \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots \neg c_m)^N := \begin{cases} a \leftarrow b_1, \ldots, b_n, & \text{if } \forall j : c_j \notin N, \\ \mathbf{t}, & \text{otherwise.} \end{cases}$$

P^N is called the *Gelfond-Lifschitz transform*. P^N is always a *definite* program. Therefore, we can compute its least Herbrand model M_{P^N} and check whether it coincides with the model N with which we started:

Definition 3 STABLE.
N is called a *stable* model of P, if $M_{PN} = N$. The stable semantics $STABLE(P)$ is given by the set of (classical or two-valued) stable models of P.

WFS maps a program into one of its three-valued Herbrand Models (due to lack of space we cannot give the full definition of WFS here). WFS relates to STABLE as follows [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.

Our earlier statement that STABLE extends WFS with respect to $\leq_k$ depends on the interpretation of $STABLE(P)$ if P has no stable models. If in analogy to classical logic we define $STABLE(P) = B_P \cup \neg B_P$ if P has no stable models, then we have $WFS \leq_k STABLE$. If we consider STABLE undefined in that case, WFS and STABLE are incomparable:

Example 3 Adding irrelevant clauses.

$$P_{stratified} : a \leftarrow \neg b \quad P_{no\_stable\_model} : \begin{array}{l} a \leftarrow \neg b \\ p \leftarrow \neg p \end{array}$$

Both WFS and STABLE coincide with the supported model on $P_{stratified}$: therefore, a is derivable. WFS also derives a from $P_{no\_stable\_model}$ which equals $P_{stratified}$ plus the clause $p \leftarrow \neg p$. In contrast, $P_{no\_stable\_model}$ does not have stable models.

3 Abstract Approach to Reasonable Semantics

All examples (1, 2, and 3) mentioned so far contain a negative cycle, the main crux of the whole enterprise of giving meaning to programs with negation. In order to cope with this problem, but not to argue on arbitrary example programs, the first author formulated abstract properties which a semantics should have.

It is possible to define the notion of a *well-behaved* semantics (see [Dix94]) as a sceptical semantics with certain such properties. Some of them state the equality between the semantics on programs which may be transformed into each other by arguably *meaning-preserving* steps. We will not review this definition here: instead, we illustrate the properties which we need here on some examples.

Look for instance at the following two (closely related) intuitive properties, which we require any reasonable semantics SEM to have:

C_1: If $a \leftarrow l_1, \ldots, l_k \in P$ and SEM(P)$\models l_1 \wedge \ldots \wedge l_k$ then SEM(P)$\models a$,
Cut: If SEM(P)$\models a$ and SEM$(P \cup \{a\}) \models x$ then SEM(P)$\models x$.

The most important principle for our present goal is the *Principle of Relevance*, which may be motivated by looking at the way a program is written down. A program as a set of clauses expresses a set of implicational dependencies between literals. While there may be mutual dependencies, the *basic structure* of a program is of a hierarchical nature. The developer of a program will deliberately choose this structure when writing the program and will most probably write down the (recursive) definition of some literals before proceeding to literals which may or may not use the program written so far.

The formalization of this idea needs to list those rules, which could ever contribute to l's derivation (or to its nonderivability by negation as failure). This is captured by the set of *relevant rules of a program P with respect to a literal l* (see Section 2). In general, l may depend on a large set of atoms. But rules that do not contain these atoms in their heads should not affect the meaning of l in P: these rules are therefore *not relevant for l*.

The *Principle of Relevance* states exactly that it is reasonable for the truth-value of a literal l only to depend on the subprogram formed from the *relevant rules* of P with respect to l.

Definition 4 Relevance.
For all literals l: $\mathrm{SEM}(P)(l) = \mathrm{SEM}(rel\_rul(P, l))(l)$.

STABLE does not satisfy this principle (see Example 3) due to the nonexistence of stable models. As to the well-founded semantics, *Relevance* holds. Equally important and intimately related is the following principle:

Definition 5 Modularity.
Let $P = P_1 \cup P_2$ and for every $a \in B_{P_2}$: $rel\_rul(P, a) \subseteq P_2$.
The principle of Modularity is: $\mathrm{SEM}(P) = \mathrm{SEM}({P_1}^{SEM(P_2)} \cup P_2)$.

P_1 can be seen "to make calls to the subprogram P_2". The assumption says that all necessary information about $a \in B_{P_2}$ is contained in P_2. Therefore, we can *reduce* the program P_1 with all literals that are already decided in P_2: ${P_1}^{SEM(P_2)}$. The formal definition of *reducing a program P with a set M of literals* cannot be given here for lack of space. Intuitively though, every occurrence in P of a literal from M is evaluated to true and false and the program P is simplified accordingly [Dix94]. This idea is present also in the context of truth maintenance systems [Wit91]. Note that *Modularity* is satisfied for STABLE.

Recently, Alferes et al. ([ADP94]) used these properties to define a *Top-Down Query Evaluation* method for an extension of WFS. Clearly, if a semantics does not satisfy *Relevance* it is not amenable to any form of Top-Down computation, simply because a query Q does not only depend from the atoms below it (i.e., from $dependencies\_of(Q)$).

The property of *Rationality* will be needed for a lemma to follow:

Rationality: If SEM(P)$\not\models \neg a$ and SEM(P)$\models x$ then $\mathrm{SEM}(P \cup \{a\}) \models x$.

The lemma presents a general construction of extending a (rational) semantics that does not satisfy *Cut* to one where the *Cut* holds.

Lemma 6 SEM$^+$.
Let SEM be a rational semantics and define an increasing sequence of sets of atoms M_i by $M_0 := \emptyset$, $M_{i+1} := SEM(P \cup M_i)$. We have

$$SEM(P \cup M_i) \leq_k SEM(P \cup M_{i+1}).$$

There exists therefore a γ such that $M_\gamma = M_{\gamma+1}$. We define

$$SEM^+(P) := SEM(P \cup M_\gamma).$$

SEM^+ is the weakest extension of SEM satisfying Cut*. In addition, SEM^+ is also rational.*

The proof is obvious: we only add those atoms that have to be added in order to satisfy *Cut*. We need *Rationality* to prove inductively $\mathrm{SEM}(P \cup M_i) \leq_k \mathrm{SEM}(P \cup M_{i+1})$[3] and to reach a limit M_γ. Otherwise, the sequence M_i could be oscillating.

The last abstract property we need is that *Supraclassicality*, well-known in the context of general nonmonotonic reasoning. Schlipf used it to overcome both the weakness of the well-founded semantics (see Example 2) and the inconsistency of the stable semantics by looking for extensions satisfying the following property:

Supraclassicality[4]*: $P \models a$ implies $a \in SEM(P)$ for atoms a.*

Supraclassicality solves some of the case splitting problems and also removes in many examples the inconsistency (e.g. in Examples 2 and 3).

4 Variants of STABLE

4.1 Two First Attempts

One might think that the inconsistency problem of STABLE can simply be solved by redefinition if no stable models exist. Taking the relation $WFS \leq_k STABLE$ as evidence, the following definition is the easiest such definition:

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

This definition has also be used in [SNV92], albeit implicitly: There, STABLE models are computed as extensions of WFS, and WFS remains as an approximation if this extension fails. STABLE* is obviously comparable to WFS. In combination with what was said earlier we have WFS $\leq_k$ STABLE* $\leq_k$ STABLE.

Unfortunately, STABLE* only solves some of the toy problems. Slightly more complex programs (e.g., $P_{splitting} \cup \{p \leftarrow \neg p\}$) show that STABLE* does not satisfy *Relevance*. Consider the following example:

[3] The definition of M_{i+1} ensures that if $x \in M_{i+1}$ then $\neg x \notin \mathrm{SEM}(P \cup M_i)$ so that *Rationality* can be applied.

[4] Schlipf calls this property *classical completeness* in [Sch92].

Example 4 (Non-) Existence of Stable Models.

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

While P_{stable} has the unique stable model $\{p, a\}$, $P_{\neg\exists\, stab.}$ fails to have any stable model. P_{stable} has the unique stable model $\{p, a\}$ and $P_{stable} \cup \{p\}$ has two stable models: $\{p, a\}$ and $\{p, b\}$.

P_{stable} has exactly the stable model $\{p, a\}$, so that a is derivable. This seems to be very strange because the relevant part of P with respect to a is just the program consisting of the first two clauses, from which nothing can be derived.[5]

Note that the supraclassicality condition already avoids the problem in Example 4: since p is derivable in P_{stable}, supraclassical semantics (like STABLE_C and STABLE^+ to be introduced below) make no difference between P_{stable} and $P_{stable} \cup \{p\}$. But more complex programs can be constructed to show the failure of *Relevance*. In addition, even if we define a semantics by *explicitly focussing on the relevant part of the program*[6] we can show that such a semantics does not satisfy the condition C_1 (nor the *Cut*) which makes it useless. We refer the reader to [DM94] for a full discussion.

4.2 Schlipf's Approach Revisited

In [Sch92] John Schlipf also noticed a serious drawback of STABLE besides the inconsistency. He defines the notion of a *stratified pair* which, interestingly enough, coincides with the condition used in *Modularity* (Definition 5):

> (P_2, P_1) is a stratified pair if for $P = P_1 \cup P_2$ the following holds:
> for every $A \in B_{P_2}$: $rel\_rul(P, A) \subseteq P_2$.

Based on this notion he define a principle of *weak stratification*:

Definition 7 (Schlipf) Weak Stratification.
Let (P_2, P_1) be a stratified pair of logic programs. The principle of *Weak Stratification* states: If a relation R appears in P_2, then $R(t) \in SEM(P_1 \cup P_2)$ **iff** $R(t) \in SEM(P_2)$.

Modularity is weaker than *Weak Stratification.* While he requires that $R(t) \in SEM(P_1 \cup P_2)$ **iff** $R(t) \in SEM(P_2)$ (*not* true for STABLE) we require that $\mathrm{SEM}(P) = \mathrm{SEM}({P_1}^{SEM(P_2)} \cup P_2)$ (satisfied by STABLE). Again *Relevance* is crucial:

Lemma 8 Modularity and Relevance imply Weak Stratification.
Any semantics satisfying Modularity and Relevance also satisfies Weak Stratification.

[5] We have two stable models: $\{a, \neg b\}$ and $\{\neg a, b\}$.
[6] Which means that *Relevance* is automatically built-in.

Therefore we think that Schlipf's notion of weak stratification is overloaded and should be split into *Modularity* and *Relevance*: The latter being the proper reason why weak stratification is not satisfied.

Schlipf introduces two semantics WFS_C (wellfounded-by-case) and STABLE_C (stable-by-case) using *derived rules*. His definitions are very technical and far from being elegant. We do not repeat his definitions here; instead, we give an elegant and useful definition of a semantics equivalent to WFS_C. The definition is based on Lemma 6 and considerably simplifies the original one from [Dix92]:

Definition 9 WFS$^+$.
Let P be a normal program, and define

$$\mathrm{SEM}(P) := \mathit{WFS}(P) \cup \{a :\ a \text{ an atom with: } P \models a\}.$$

The assumptions of Lemma 6 are satisfied and we get a rational semantics WFS$^+$.

Theorem 10 WFS$^+$ as $\leq_k$-weakest extension of WFS.
WFS$^+$ is the weakest extension of WFS that satisfies Cut *and* Supraclassicality*. In addition WFS$^+$ is rational and satisfies the following property (stronger than Supraclassicality):*

$$\textit{If } P \cup \mathit{WFS}^+(P) \models a \textit{ then } a \in \mathit{WFS}^+(P).$$

Using the last theorem it is easy to prove that Schlipf's WFS_C essentially coincides with WFS$^+$ for all programs. A straightforward attempt to incorporate Supraclassicality explicitly into the definition of STABLE gives the following:

Definition 11 STABLE$^{\#}$.

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

STABLE$^{\#}$ is supraclassical and satisfies *Cut*. It does not satisfy the strong version of Supraclassicality.[7]

Iterating the construction of STABLE$^{\#}$ gives us a semantics STABLE$^+$ which is strongly related to STABLE_C:

Definition 12 STABLE$^+$.
Let P be a normal program. If $P \cup \{a :\ (P \cup \mathit{WFS}^+(P)) \models a\}$ has no stable models, set $\mathit{STABLE}^+(P) := \mathit{WFS}^+(P)$. Otherwise, define a decreasing sequence M_i of sets by $M_0 := \mathit{STABLE}(P \cup \{a :\ (P \cup \mathit{WFS}^+(P)) \models a\})$ and

$$M_{i+1} := \mathit{STABLE}(P \cup \{a :\ (P \cup M_i) \models a)\}.$$

We define $$\mathit{STABLE}^+(P) := \textstyle\bigcap_i M_i.$$

[7] I.e., the appropriate instance of "If $P \cup \mathit{SEM}(P) \models a$ then $a \in \mathit{SEM}(P)$

Lemma 13 STABLE$^+$ $\leq_k$ STABLE.
STABLE$^+$ is a semantics below STABLE$^\#$ (and therefore also below STABLE) satisfying Cut *and* Strong Supraclassicality*. On all programs where $STABLE_C(P)$ is consistent, $STABLE_C(P)$ =$STABLE^+(P)$.*

The next example shows that all these semantics are no solutions to *Relevance*. In addition they are strict extensions with respect to $\leq_k$:

Example 5 Properties of $STABLE_C$, $STABLE^+$, $STABLE^\#$.

$$P_{Schlipf1} : \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg p \\ a \leftarrow p, \neg b \\ b \leftarrow p, \neg c \\ c \leftarrow p, \neg a \end{array} \qquad P_{Schlipf2} : \begin{array}{l} a \leftarrow \neg b \\ c \leftarrow a, \neg d \\ d \leftarrow a, \neg c \\ e \leftarrow a, \neg e \\ e \leftarrow a, c \end{array}$$

$P_{Schlipf1}$ has only one stable-by-case model: $\{\neg p, q, \neg a, \neg b, \neg c\}$. But $rel\_rul(P_{Schlipf1}, p) = rel\_rul(P_{Schlipf1}, q) = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$. Concerning $P_{Schlipf2}$: no atoms can be derived classically. Hence, STABLE$^\#$ coincides with STABLE and derives $\{a, c, e, \neg b, \neg d\}$. STABLE$_C$ only derives a and e.

We therefore have: WFS$^+$ $\leq_k$ STABLE$^+$ $\leq_k$ STABLE$^\#$ $\leq_k$ STABLE.

4.3 Other Variants

Closely related to STABLE is REG-SEM [YY93], the semantics based on regular models. You and Yuan showed [YY93] that REG-SEM is equivalent to

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

These semantics satisfy *Relevance*, *Modularity*, *Cut* but not *Rationality* ([DM94]) and therefore can also be seen as interesting *well-behaved* approximations of STABLE. In addition, REG-SEM is stronger than WFS$^+$:

Example 6 REG-SEM vs. WFS$^+$.

$$P_{splitting} : \begin{array}{l} a \leftarrow \neg b \\ b \leftarrow \neg a \\ p \leftarrow a \\ p \leftarrow b \end{array} \quad P'_{splitting} : \begin{array}{l} a \leftarrow \neg b \\ b \leftarrow \neg a \\ p \leftarrow \neg b \\ p \leftarrow \neg a \end{array}$$

The second program results from the first by evaluating a (by $\neg b$) and b (by $\neg a$). While WFS$^+$ derives p from $P_{splitting}$, it does not derive p from $P'_{splitting}$. REG-SEM derives p from both programs!

Note that REG-SEM is not supraclassical: for "$p \leftarrow \neg p$", the atom p is assigned undefined. Thus, REG-SEM is incomparable with WFS$^+$. But it is possible to define (in much the same way as WFS$^+$) a semantics WFS$'$ that shares all nice properties with WFS$^+$, except the supraclassicality. These semantics are related as follows:

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

In [DM94] we show how REG-SEM is related to *unfolding*.

5 Conclusion and related attempts

We have shown that a major reason of certain anomalies of the STABLE semantics is its failure of *Relevance*. This property is strongly related to Schlipf's notion of *weak stratification* and essential for Top-Down Query Evaluation methods. This has also been observed in [ADP94], where our abstract approach is used to implement an extension of WFS containing explicit negation. Lifschitz and Turner [LT94] investigate a notion closely related to *Relevance*: *splitting a program*. A set U is a *splitting set*, if U contains with every atom a also all literals of $rel\_rul(P, a)$. Brewka [Bre93] approached the anomalies of STABLE by considering abduction but his proposal did not completely capture negation as failure. In his framework it is possible that atoms occuring in no head are assigned *undefined* instead of *false*.

In addition, we discussed extensions of WFS which allow for reasoning-by-cases and can be seen as *well-behaved* approximations and improvements of STABLE. Our approach enabled us to give an elegant definition of Dix' WFS$^+$ and to show that it coincides with Schlipf's WFS$_C$. We also noted that REG-SEM is a relevant approximation of STABLE.

6 Acknowledgements

The authors would like to thank John Schlipf for very helpful remarks and clarifying examples. His suggestions helped to improve the paper a lot. Two anonymous referees also provided us with useful suggestions which, along with the space restrictions, led us to a major rewriting of this paper.

References

[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.

[ADP94] J. J. Alferes, Carlos Viegas Damasio, and L. M. Pereira. Top-down query evaluation for well-founded semantics with explicit negation. In *Proceedings ECAI*. John Wiley & Sons, 1994.

[BF91] Nicole Bidoit and Christine Froidevaux. General logical Databases and Programs: Default Logic Semantics and Stratification. *Information and Computation*, 91:15–54, 1991.

[Bre93] Gerhard Brewka. An Abductive Framework for Generalized Logic Programs. In *Logic Programming and Non-Monotonic Reasoning, Proc. 2nd Int. Workshop*, MIT Press, July 1993.

[Dix91] Jürgen Dix. Classifying Semantics of Logic Programs. In *Logic Programming and Non-Monotonic Reasoning, Proc. 1st Int. Workshop*, pages 166–180. MIT Press, July 1991.

[Dix92] Jürgen Dix. A Framework for Representing and Characterizing Semantics of Logic Programs. In *Principles of Knowledge Representation and Reasoning: Proc. 3rd Int. Conf.*, pages 591–602. Morgan Kaufmann, 1992.

[Dix94] Jürgen Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In *Logic, Action and Information. Proc. Konstanz Colloquium in Logic and Information (LogIn '92).* DeGruyter, 1994.

[DM94] Jürgen Dix and Martin Müller. Extensions and Improvements of the Stable Semantics. In *Proc. 8th Int. Symp. on Methodologies for Intelligent Systems.* Springer, LNAI, to appear, 1994.

[Dun91] P. M. Dung. Negation as Hypotheses: An Abductive Foundation for Logic Programming. In *Proc. 8th Int. Conf. on Logic Programming.* MIT Press, June 1991.

[FLMS93] J. A. Fernandez, J. Lobo, J. Minker, and V. S. Subrahmanian. Disjunctive LP + Integrity Constraints = Stable Model Semantics. *Annals of Mathematics and Artificial Intelligence*, 8(3-4), 1993.

[GL88] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In *5th Int. Conf. on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[LT94] Vladimir Lifschitz and Hudson Turner. Splitting a Logic Program. This volume.

[MS89] Wiktor Marek and V.S. Subrahmanian. The Relationship between Logic Program Semantics and Nonmonotonic Reasoning. In *Proc. 6. Int. Conf on Logic Programming*, pages 600–617. MIT Press, July 1989.

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

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

[SNV92] V.S. Subrahmanian, Nau, and Vago. WFS + Branch and Bound = Stable Models. Technical Report CS-TR-2935, Univ. of Maryland, July 1992. To be published with *IEEE Transactions on Knowledge and Data Engineering*

[SZ91] D. Sacca and C. Zaniolo. Partial models and Three-Valued Models in Logic Programs with Negation. In *Logic Programming and Non-Monotonic Reasoning, Proc. 1st Int. Workshop*, pages 87–104. MIT Press, July 1991.

[Tor93] Alberto Torres. Negation as Failure to Support. In *Logic Programming and Non-Monotonic Reasoning, Proc. 2nd Int. Workshop*, pages 223–243. MIT Press, July 1993.

[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.

[Wit91] Cees Witteveen. Partial Semantics for Truth Maintenance. In *Logics in AI.* Springer, LNAI 478, 1991.

[YY93] Jia-Huai You and Li-Yan Yuan. On the Equivalence of Semantics for Normal Logic Programs. In *Proc. Workshop on Logic Programming with Incomplete Information, following ILPS' 93*, pages 161–176, 1993.

This article was processed using the LaTeX macro package with LLNCS style