Partial Evaluation and Relevance for Approximations of the Stable Semantics

Jürgen Dix[1] and Martin Müller[2]

[1] University of Koblenz, Dept. of Computer Science,
Rheinau 1, D-56075 Koblenz, Germany
[2] Graduiertenkolleg Kognition and German Research Center for AI (DFKI),
Stuhlsatzenhausweg 3, D-66123 Saarbrücken

Abstract. There is no doubt that STABLE and WFS are among the dominant semantics for logic programs. While WFS has many nice structural properties, it is very weak. STABLE allows to derive more atoms, but may become inconsistent and it is not relevant, i.e. the truth value of an atom does not only depend on the call-graph below it. In this paper we consider the problem of defining approximations of STABLE that are both *relevant* and satisfy a general *partial deduction* property.

1 Introduction and Motivation

The well-founded semantics WFS [vGRS91] and the stable semantics STABLE [GL88] are undoubtedly the dominant semantics for normal logic programs. Both of them underly very simple intuitions: The well-founded *three-valued* model is in a sense the *weakest reasonable* semantics defined on *all* normal programs ([Dix92]). STABLE, on the other hand, is defined by a set of *two-valued* models which can therefore be used for both *sceptical* and *credulous* reasoning by considering *all* or *some* of the stable models for entailment. An additional, distinguishing feature of STABLE is that every stable model can be seen as a solution of the constraints expressed by the program. Such a solution corresponds to a complete (two-valued) scenario (the "scenario view" of a semantics [DM94]). The shortcomings of both semantics are also well-known: WFS is often weaker than intended since it prohibits case-based reasoning, and STABLE may become inconsistent. Moreover, the STABLE semantics does not allow top-down query processing, because the answer to a query does not only depend on its call-graph, i.e. the subprogram that can be recursively reached by tracing program clauses. Consider the well-known examples below: p is not in WFS(P_{split}), although $a \vee b$ and p hold in every two-valued model of P_{split}. STABLE(P_{split}) derives p while WFS(P_{split}) leaves p undefined.

Example 1 WFS vs. STABLE.

$$
\begin{array}{llll}
P_{split}: & a \leftarrow \neg b & \qquad P_{no\text{-}stable}: & p \leftarrow \neg p \\
 & b \leftarrow \neg a & & a \\
 & p \leftarrow a & & \\
 & p \leftarrow b & &
\end{array}
$$

WFS behaves very regularly as indicated by many structural properties. In particular, it satisfies a very general principle of *partial evaluation* and it is *relevant*: Notions to be defined below. The well-founded model is an approximation of the sceptical semantics induced by the stable models. Some implementations of STABLE use this fact explicitly [SNV92] and some do not [CW93]. In this paper we consider the problem of closing the gap between WFS and STABLE by investigating extensions of WFS towards STABLE which still share the well-behavedness of WFS.

This goal is approached in two ways: The first singles out the *regular model semantics* REG-SEM [YY93] as such a well-behaved approximation. Since REG-SEM is given by a set of (three-valued) models, it still allows for scenario reasoning where the scenarios may become incomplete, though. The second one enriches the definition of STABLE by imposing certain properties. We develop a semantics STABLE^{rel} by trying to stick as close as possible with the spirit of (two-valued) STABLE and yet to avoid the inconsistency problem. STABLE^{rel} has a more constructive flavor.

The plan of the paper is as follows: In Section 2 below, we give some notation. In Section 3 we review the stable, the well-founded, and the regular semantics along with their relationship. Section 4 briefly describes the abstract properties we consider in this paper: *Cut*, *Relevance*, *PPE*, and *GPPE*. In Section 5 we use our approach to define a relevant version of STABLE: STABLE^{rel}. We end in Section 6 with some remarks about disjunctive logic programs, related attempts and conclude.

2 Notation

A program clause or a *rule* is a formula

$$a \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots \neg c_l, \text{ where } n \geq 1 \text{ and } l \geq 0.$$

As usual, the comma represents conjunction. A *rule* is called *definite* 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 here we restrict ourselves to fully grounded programs. Furthermore, we spare to discuss infinite such programs by restriction to programs with the bounded term size property (e.g. *datalog* programs). In the sequel *program* will mean *fully grounded program* and its finiteness is to be understood.

We will also use the truth values $\mathbf{t}$, $\mathbf{u}$, and $\mathbf{f}$ which must be assigned the truth values *true*, *unknown*, or *false* in every model. We call $\mathbf{u}$-*programs* those programs that contain an occurrence of $\mathbf{u}$ in their bodies.

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

- $dependencies\_of(x) := \{a : x \text{ depends on } a\}$,
- $rel\_rul(P, x)$ 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 head,

These definitions are straightforwardly extended to arbitrary literals by setting $dependencies\_of(\neg x) := dependencies\_of(x)$ and $rel\_rul(P, \neg x) := rel\_rul(P, x)$. Our notion of a semantics is a very general one [Dix91, Dix92]:

Definition 1 SEM.
A semantics SEM is a mapping from the class of all programs into the powerset of the set of all three-valued Herbrand Structures. SEM assigns to every program P a set of three-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:

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

As our main emphasis lies on the sceptical semantics, we will henceforth use SEM instead of SEM^{scept}. We distinguish two 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, and
- 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 SEM and SEM' coincide.

As an example, STABLE extends WFS in the former sense (see Definition 4) while both STABLE and WFS extend the least Herbrand model semantics for *definite* or the least supported semantics for *stratified* programs in the latter sense. The notation $\leq_k$ comes from the partial ordering $\mathbf{u} \leq_k \mathbf{t}$ and $\mathbf{u} \leq_k \mathbf{f}$ (*knowledge* ordering, hence k). Minimal models are defined via the *truth* ordering $\leq_t$ defined by $\mathbf{f} \leq_t \mathbf{u} \leq_t \mathbf{t}$.

3 Stable, Well-founded, and Regular 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]. We follow the presentation of [Prz90]:

Definition 3 Extended Gelfond-Lifschitz Transformation.
Let P be a program and $N \subseteq B_P$ be a three-valued model of P, then the *extended Gelfond-Lifschitz transformation* (or simply GL transformation) P^N is obtained from P by replacing every *negative* body literal $\neg c$ in P by $\mathbf{f}$ (resp. $\mathbf{u}$ or $\mathbf{t}$) if N assigns c the truth value $\mathbf{t}$ (resp. $\mathbf{u}$ or $\mathbf{f}$). Furthermore, a clause with $\mathbf{f}$ in its body can be erased, and $\mathbf{t}$ can be dropped from the body of every clause.

Note that P^N is always a *definite* $\mathbf{u}$-program which has a unique $\leq_t$-minimal (three valued) Herbrand Model M_{P^N} ([Prz90, Proposition 3.1.]). Comparison of N and M_{P^N} answers whether N is a stable model:

Definition 4 STABLE, PART-STABLE, and REG-SEM.
N is called a *stable* model of P, if $M_{P^N} = N$ and N is two-valued. It is called a *partial stable* model if N is three-valued. The stable semantics STABLE(P) is given by the set of two-valued partial stable models, PART-STABLE(P) by the set of all, possibly three-valued, stable models of P. REG-SEM(P) is given by the set of $\leq_k$-maximal partial stable models of P: We call them *regular* models[3].

The well-founded semantics relates to STABLE as follows [vGRS91, Prz90]:

- WFS(P) coincides with PART-STABLE(P) (viewed sceptically).
- Every stable model N of P extends WFS(P): WFS(P) $\leq_k N$.
- If WFS(P) is two-valued, WFS(P) is the unique stable model.

The relationship between REG-SEM and a number of other semantics is discussed in [YY93] (see also [DM94]). Of course, WFS $\leq_k$ REG-SEM $\leq_k$ STABLE holds. For illustration consider the following example:

Example 2 REG-SEM vs. STABLE.

$$P_{\neg\exists\, stab.}: \begin{array}{l} a \leftarrow \neg b \\ b \leftarrow \neg a \\ p \leftarrow \neg p \end{array} \qquad P_{regsem}: \begin{array}{l} a \leftarrow \neg b \\ b \leftarrow \neg a \\ p \leftarrow \neg a \\ p \leftarrow \neg p \end{array}$$

$P_{\neg\exists stab.}$ has no stable model, but two regular ones: $\{a, \neg b\}$, and $\{b, \neg a\}$. P_{regsem} has the unique stable model $\{p, b, \neg a\}$, but it has two regular models, namely $\{p, b, \neg a\}$ and $\{a, \neg b\}$.

4 Abstract Approach to Reasonable Semantics

Many semantics have been defined in order to give meaning to logic programs with negation. The abstract approach to semantics has been introduced in [Dix92] in order to compare and evaluate semantics against each other and to avoid arguments based solely on small example programs. It has also been employed to reconstruct semantics based on intended structural properties. In this section we give the definition of only those principles which will be needed below.

[3] They were originally defined in [YY90].

4.1 Relevance and Cut

First consider the following two (closely related) properties, which we require any reasonable semantics SEM to have [Dix92]:

> 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)$\models a$, and
> *Cut*: If SEM(P)$\models a$ and SEM($P \cup \{a\}$) $\models x$ then SEM(P)$\models x$.

Most important for our present goal is the *Principle of Relevance*. For motivation, consider the development of a program as a set of implicational dependencies between literals. Whilst there may be mutual dependencies, the *basic structure* of a program is of a hierarchical nature which the programmer deliberately chooses: She will most probably write down the (recursive) definition of some literals before proceeding to others which may or may not use the program written so far. The *Principle of Relevance* states that the evaluation of a semantics has to respect these dependencies: The truthvalue of a literal l should only depend on the *relevant rules* of P with respect to l.

Definition 5 Relevance.
For all literals l: SEM$(P)(l)$ = SEM$(rel_rul(P,l))(l)$.

Relevant semantics, for instance WFS, allow *top-down query evaluation*. This has also been noted for an extension of WFS by Alferes et al. [ADP94]. Non-relevant semantics are not amenable to any form of top-down computation because they employ a global argument (a query Q may depend on atoms not in $dependencies\_of(Q)$). The latter applies to STABLE and is related to the nonexistence of stable models for programs such as $P_{no\text{-}stable}$ in Example 1: STABLE does not satisfy Relevance.

4.2 Variants of Partial Evaluation

Partial evaluation principles allow to transform a program into another one from which the semantics may be more efficiently computed, or which has a more regular structure. We consider two reasonable variants:

Definition 6 Strong PPE.
Let P be a program such that the atom c occurs only positively in P. Let $c \leftarrow rhs_1,\ \ldots\ ,\ c \leftarrow rhs_n$ be all the rules of P with head c. Define P' to be a program obtained from P by replacing some $a \leftarrow c, body \in P$ by the rules

$$a \leftarrow rhs_1, body, \quad \ldots \quad a \leftarrow rhs_n, body$$

Note that the rules $c \leftarrow rhs_1,\ \ldots\ , c \leftarrow rhs_n$ are not removed (in contrast to the weak PPE [Dix92]). The *principle of partial evaluation* is: $SEM(P') = SEM(P)$.

Definition 7 GPPE.
The *generalized* principle of partial evaluation is obtained from PPE by dropping the assumption that c only occurs positively in P.

One central difference between PPE and GPPE is that stratifiability of a disjunctive program is invariant under PPE but not under GPPE. PPE is more syntax-sensitive so to speak. As an example for a semantics satisfying PPE but not GPPE, consider WFS_C [Sch92] (or, equivalently, WFS^+ [Dix92]):

Example 3 Generalized PPE fails for WFS_C.

$$
\begin{array}{llll}
P_{split}: & a \leftarrow \neg b & P'_{split}: & a \leftarrow \neg b \\
 & b \leftarrow \neg a & & b \leftarrow \neg a \\
 & p \leftarrow a & & p \leftarrow \neg b \\
 & p \leftarrow b & & p \leftarrow \neg a
\end{array}
$$

P'_{split} results from P_{split} by evaluating a (by $\neg b$) and evaluating b (by $\neg a$). WFS_C derives p from P_{split}, but it does not derive p from P'_{split}. Note that if enforce GPPE (i.e., derive p from P'_{split} by declaration) then *Rationality* ([Dix91]) fails: $\mathrm{WFS}_C(P'_{split})$ contains neither $\neg a$ nor $\neg b$. Yet, adding a and b does not only inhibit the derivation of p, but makes even $\neg p$ derivable!

REG-SEM derives p from P'_{split}. Indeed, GPPE holds for REG-SEM:

Theorem 8 GPPE holds for WFS, REG-SEM and STABLE.
WFS, REG-SEM and STABLE satisfy GPPE.

The fact that only *positive* occurrences of c are replaced, makes the proof obvious for WFS. For REG-SEM and STABLE, we observe that GPPE and the GL transformation commute, and that the following lemma holds:

Lemma 9 Positive u-programs are invariant under GPPE.
Let P be a positive **u***-program and P' the result of applying GPPE to P. Then P and P' have the same $\leq_t$-minimal models.*

5 STABLE^{rel}

In this section we develop STABLE^{rel} in order to approximate the twovalued stable semantics but to avoid the inconsistency problem. At the first glance, it might seem that the solution is a redefinition of STABLE in case no stable models exist. Recalling that $WFS \leq_k STABLE$ holds, we define:

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

and observe that WFS $\leq_k$ STABLE* $\leq_k$ STABLE. This could be seen to be the semantics underlying the algorithm in [SNV92], where STABLE is computed but WFS remains as an approximation if there are no stable models. Already he program $P_{split} \cup \{p \leftarrow \neg p\}$ shows that STABLE* does not satisfy *Relevance*. But the following example illustrates that the inconsistency is obviously not the only problem.

Example 4 (Non-) Existence of Stable Models.

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

P_{stable} has the unique stable model $\{p, a\}$, but $P_{\neg\exists\, stab.}$ has no stable model. P_{stable} has the unique stable model $\{p, a\}$ and $P_{stable} \cup \{p\}$ has two stable models: $\{p, a\}$ and $\{p, b\}$.

From the single stable model of P_{stable} we derive a, but the relevant part of P with respect to a is just the program consisting of the first two clauses which have two stable models $\{a, \neg b\}$ and $\{\neg a, b\}$ and disallow the derivation of a.

We expect a reasonable semantics *not to derive a from P_{stable}*. Therefore, the semantics we are looking for cannot be an extension of STABLE*. Moreover:

Lemma 10. *There is no semantics SEM that coincides with STABLE, if stable models exist ($SEM^{scept}(P) = STABLE^{scept}(P)$), and satisfies* Relevance.

This lemma shows that the variant of the stable semantics recently defined by Sacca (the least undefined stable model from [Sac93]) is not relevant.

We now define a version of STABLE which is relevant. Reconsider Example 4 and use it positively. It suggests a natural approximation: We enforce *Relevance* by just *focussing on the relevant part* of a program in the redefinition of STABLE. This idea is captured by STABLE**:

$$\mathrm{STABLE}^{**}(P)(l) := \begin{cases} \mathrm{STABLE}(\mathit{rel\_rul}(P, l))(l), & \text{if stable models exist,} \\ \mathrm{WFS}(\mathit{rel\_rul}(P, l))(l), & \text{otherwise.} \end{cases}$$

Although this version does satisfy *Relevance*, it has another serious flaw:

Example 5 STABLE**.

$$\begin{array}{lll} P_{STABLE^{**}}: & a \leftarrow \neg b & x \leftarrow \neg x, a \\ & b \leftarrow \neg a & y \leftarrow \neg y, b \\ & & w \leftarrow \neg x, \neg y. \end{array}$$

Note that according to our definition of STABLE**, both $\neg x$ and $\neg y$ are derivable but w is not: The most natural condition C_1 (see Section 4) is not satisfied. Similar counterexamples show that *Cut* also does not hold.

The fall back to WFS may seem ad-hoc, but is the weakest possibility [Dix92]. The construction to follow, which repairs the failure of Relevance, does not really depend on WFS but can be parametrized by stronger semantics, too.

Example 5 indicates that we need a *recursive* definition which will yield STABLE^{rel}. First, we define an equivalence relation $\sim$ between atoms of B_P:

$$a \sim b \quad \underline{:\text{iff}} \quad (a \text{ depends on } b \text{ and } b \text{ depends on } a) \text{ or } a = b.$$

Second, we take $\preceq$ to be the partial order induced by $\sim$ on $\sim$-equivalence classes:

$$\overline{a} \preceq \overline{b} \quad \underline{:\text{iff}} \quad b \text{ depends on } a.$$

Call an atom to be *of order 0*, if $\overline{a}$ is minimal in $\preceq$. If, for some atom a, the maximal order of the atoms of which a depends is n, then a is called to be *of order* $n+1$. Since we only consider finite ground programs, all atoms get a finite order. We extend this notion to literals in the obvious way: A literal $\neg l$ is of order n iff l is.

Definition 11 STABLE^{rel}.
For literals l of order 0 we set

$$\mathrm{STABLE}_0^{rel}(P)(l) := \begin{cases} \mathrm{STABLE}(\mathit{rel\_rul}(P,l))(l), & \text{if stable models exist,} \\ \mathrm{WFS}(\mathit{rel\_rul}(P,l))(l), & \text{otherwise,} \end{cases}$$

and for literals l of order $n+1$

$$\mathrm{STABLE}_{n+1}^{rel}(P)(l) := \begin{cases} \mathrm{STABLE}(\mathit{rel\_rul}(P,l)^{STABLE_n^{rel}(P)})(l), & \text{if } \exists \text{ stab.mod.} \\ \mathrm{WFS}(\mathit{rel\_rul}(P,l)^{STABLE_n^{rel}(P)})(l), & \text{otherwise.} \end{cases}$$

That is, we reduce $\mathit{rel\_rul}(P,l))$ with all literals that have been evaluated to *true* or *false* in a previous step and then apply STABLE or WFS, respectively. According to STABLE^{rel} we now do derive w from $P_{STABLE^{**}}$ (Ex. 5).

The formal definition of the reduction $P^{\langle T;F \rangle}$ of a program P with respect to a three-valued model (see [Dix92]) is omitted due to space limitations. It is important to note that $P^{\langle T;F \rangle}$ is *not* the GL-transformation. It means that all occurrences of literals in P that are assigned a $\mathbf{t}$ or $\mathbf{f}$ by $\langle T;F \rangle$ are replaced by their truth-value and P is simplified accordingly (cf. Def. 3).

Note that if at some point in the computation, at order n, say, we are forced to switch to WFS because no stable models exist, this applies to all orders $m > n$. This follows as a corollary from the following lemma.

Lemma 12.
Let $P = P_1 \cup P_2$ such that for all $a \in B_{P_2}$: $\mathit{rel\_rul}(P,a) \subseteq P_2$. Then for all stable models $\mathcal{M}$ of P: $\mathcal{M}$ restricted to B_{P_2} is a stable model of P_2.

The proof is by induction on the definition of the Gelfond-Lifschitz transform.

Let us add some words to the problem of defining STABLE^{rel} for general programs, i.e. for infinite ground programs. Here we also have to consider literals of *infinite* order. This can be done by associating arbitrary ordinals to literals and defining STABLE$_\alpha^{rel}$ accordingly. Some literals will be assigned a limit ordinal then. Note that literals may not only depend on other literals of infinite order, but also on literals of arbitrarily high finite order. Take for example the program consisting of $P(f(x)) \leftarrow P(x)$ and $Q(a) \leftarrow P(x)$. $Q(a)$ depends on $P(f^n(a))$ for all n! Finally, observe that $\preceq$ is not well-founded. The program $P(x) \leftarrow P(f(x))$ together with a constant a gives an infinitely descending sequence $P(a), P(f(a)), \ldots$: This is an illustration that not every atom can be associated a *finite* order

Constructively, we feel that STABLE^{rel} captures the spirit of STABLE, while it may, for programs with a well-founded dependency graph, be computed along the structure of the program which seems to be close to the programmers intent.

Lemma 12 also shows that STABLE^{rel} is weaker than STABLE, and the more so the higher we ascend in the order of literals: This is, because not all stable models of clauses of order n or less are necessarily stable models of clauses of order $n+1$ or less. In particular we have

$$\text{WFS} \leq_k \text{STABLE}^{**} \leq_k \text{STABLE}^{rel} \leq_k \text{STABLE}.$$

In addition, STABLE^{rel} is well-behaved:

Lemma 13.
$STABLE^{rel}$ satisfies GPPE and Relevance.

A final remark: It is possible to modify the definition of STABLE towards the scenario view by approximating models. In this case, one might compare the number of models of the subprogram $P^n \subseteq P$ of rules up to order n to those of $P^{n+1} \subseteq P$: If there are less models, then the definition of some literal of order $n+1$ must have ruled out one of the models. A contradiction even must have appeared at that level, where there are no more STABLE models at all, which can be returned to the programmer as an additional information.

6 Conclusion and Related Attempts

Consideration of disjunctive logic programs introduces more serious difficulties. It is easy to formulate GPPE for disjunctive programs: This has been done independently by Brass/Dix [BD94] and Sakama/Seki [SS94]. In both papers it is proved that DSTABLE (and GCWA) satisfy GPPE. However, DSTABLE inherits the failure of *Relevance* from its non-disjunctive version STABLE. We can now try to extend REG-SEM to disjunctive programs: Indeed, this can be done in a straightforward fashion. Unfortunately, GPPE does no longer hold for such a version DREG-SEM: Stefan Brass found a counterexample already for positive programs. But the semantics D-WFS constructed in [BD94] is relevant and satisfies GPPE (as well as PERFECT).

Concerning normal, i.e. non-disjunctive programs, we considered in a companion paper ([DM94]) the semantics WFS_C and WFS^+ and showed how they relate to STABLE. Although they extend WFS, approximate STABLE and are relevant, they do not have the general partial deduction property (GPPE). In this paper we showed that REG-SEM is a better approximation. We also defined a semantics which is even closer to STABLE and still satisfies both properties. The overall construction of STABLE^{rel} is procedural in nature and seems to be better suited for Top-Down Query Evaluation techniques.

Let us finally note that the semantics of Minker and his group (DWFS, GDWFS and their extensions) do not even satisfy our weak version of PPE and therefore are not well-behaved ([Dix91]).

7 Acknowledgements

The authors would like to thank two anonymous referees for very helpful remarks. Their suggestions helped to improve the paper a lot.

References

[ADP94] J. J. Alferes, C. V. Damasio, and L. M. Pereira. Top-down Query Evaluation for Well-founded Semantics with Explicit Negation. *Proc. ECAI.* 1994.

[BD94] S. Brass and J. Dix. A Disjunctive Semantics based on Bottom-Up Evaluation and Unfolding. *13th World Computer Congress IFIP, GI-Workshop W2 (Disjunctive Logic Programming and Disjunctive Databases)*, 1994.

[CW93] W. Chen and D. S. Warren. Computation of Stable Models and its Integration with Logical Query Processing. Technical report, SUNY at Stony Brook, 1993.

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

[Dix91] J. Dix. Classifying Semantics of Logic Programs. *Logic Programming and Non-Monotonic Reasoning, Proc. 1st Int. Workshop*, pp. 166–180. 1991.

[Dix92] J. Dix. A Framework for Representing and Characterizing Semantics of Logic Programs. *Principles of Knowledge Representation and Reasoning: Proc. 3rd Int. Conf. (KR '92)*, pp. 591–602. 1992.

[Dix94] J. Dix. A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties. *Fundamenta Informaticae*, forthcoming, 1994.

[DM94] J. Dix and M. Müller. The Stable Semantics and its Variants: A Comparison of Recent Approaches. *Proc. 18th German Annual Conf. on Artificial Intelligence (KI '94)*. Springer, LNAI, to appear, 1994.

[GL88] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. *5th Int. Conf. on Logic Programming*, pp. 1070-1080, 1988.

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

[Sac93] D. Sacca. The Expressive Power of Stable Models For DATALOG Queries with Negation. *Proc. Workshop on Logic Programming with Incomplete Information after ILPS' 93*, pp. 150–162, 1993.

[Sch92] J. 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*

[SS94] C. Sakama and H. Seki. Partial Deduction of Disjunctive Logic Programs: A Declarative Approach. *Proc. 4th Int. Workshop on Logic Program Synthesis and Transformation (LOPSTR'94)*. Springer, LNCS, 1994.

[vGRS91] A. van Gelder, K. Ross, and J. 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] J.-H. You and L.-Y. Yuan. On the Equivalence of Semantics for Normal Logic Programs. *Proc. Workshop on Logic Programming with Incomplete Information, following ILPS' 93*, pp. 161–176, 1993.

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