Revised Stable Models - a new semantics for logic programs

Luís Moniz Pereira and **Alexandre Miguel Pinto**

Centro de Inteligência Artificial, Universidade Nova de Lisboa
2829-516 Caparica, Portugal
{lmp|amp}@di.fct.unl.pt

Abstract

This paper introduces an original 2-valued semantics for Normal Logic Programs (NLP), important on its own. Nevertheless, its name draws attention to that it is inspired by and generalizes Stable Model semantics (SM). The definitional distinction consists in the revision of one feature of SM, namely its treatment of odd loops over default negation. This single revised aspect, addressed by means of a *Reductio ad Absurdum* approach, affords us a fruitful cornucopia of consequences, namely regarding existence, relevance and top-down querying, cumulativity, and implementation.

The paper motivates and then defines the Revised Stable Models semantics (rSM), justifying the definition and providing examples. It also presents two rSM semantics preserving program transformations into NLP without odd loops. Properties of rSM are given and contrasted with those of SM. Implementation is examined, and extensions of rSM are given with regard to explicit negation, 'not's in heads, and contradiction removal. Conclusions, further work, as well as potential use, terminate the paper.

Keywords: Logic Program semantics, Stable Models, *Reductio ad Absurdum*.

Introduction

This paper introduces a new 2-valued semantics for Normal Logic Programs (NLP), called Revised Stable Models semantics (rSM), cogent in its own original way. Nevertheless, its name intends to draw attention to that it is both inspired by and generalizes Stable Model semantics (SM) [4].And indeed SM models are just particular rSM models, and the SM definition a particular case of the rSM one. But its name also intends to draw attention to that the definitional distinction consists in the revision of one feature of SM, namely its treatment of odd loops over default negation. This single revised aspect affords us a fruitful cornucopia of consequences, not shared by SM, the 'de facto' standard two-valued semantics for NLP.

For one, rSM models are guaranteed to exist for every NLP, and this is important for program composition and updating, with knowledge originating in several sources. Two, rSM is relevant, meaning that there exist top-down, program call-graph based, query driven methods to determine whether a literal belongs to a model. They can thus return simply an extendable partial model, there being no need to compute all models or complete models to answer a query. Relevance is also crucial for abduction, it being query driven. Three, rSM is cumulative, so that lemmas may be stored and reused.

These and other properties, and their implementational impact, shall be examined in the sequel. Moreover, two semantics-preserving transformations are provided for NLP, so that the SMs of the transformed program correspond exactly to the rSMs of the original one. Such transformations accrue additional insight into rSM. One of them offers a vessel for immediate implementation in existing (though restrictive) SM systems.

However, another approach to implementation is possible: the top-down querying ability means that no global computing of the model is needed, and thus that no prior program grounding is required, as is the case in SM systems. Grounding is a problem in present SM implementations for writing meta-interpreters, because all possible clause bodies have to be grounded, a practical impossibility. Nevertheless, meta-interpreters are query driven, and so do not need all possible bodies to be ground.

We shall discuss an implementation avenue relying on an adaptation of ABDUAL [3], an XSB-Prolog implemented procedure that can compute Generalized Stable Models, by viewing default negated literals as abducibles, and permitting top-down querying. This bridge, fostered by rSM, brings closer together the SM based and the WFS based research communities.

Odd Loops Over Negation

In SM, programs such a ← ~a , where '~' stands for default negation, do not have a model. One can easily perceive that the Odd Loop Over Negation is the trouble-maker. The single rSM model however is {a}. The reason is that if assuming '~a' leads to an inconsistency, namely by implying 'a', then in a 2-valued semantics 'a' should be true instead.

Example 1: The president of Morelandia is considering invading another country. He reasons thus: if I do not invade them they are sure to develop Weapons of Mass Destruction (WMD); on the other hand, if they have WMD I should invade them. This is coded by his analysts as:

WMD ← ~ invade invade ← WMD

Under the SM semantics this program has no models. Under the rSM semantics invasion is warranted by the single model M={invade}, and no WMD exist.

In a NLP, we say we have a *loop* when there is a rule dependency call-graph path that has the same literal in two different positions along the path – meaning that the literal depends on itself. An Odd Loop Over Negation is one such that the number of default negations in the rule dependency graph path connecting the same literal at both ends is odd.

It is an apparently a counter-intuitive idea to permit such loops to support a literal's value of true, because it means that the truth of a literal is being supported on its negation, which seems self-inconsistent. SM does not go a long way in treating odd loops. It simply decrees there is no model (throwing out the baby along with the bath water), instead of opting for taking the next logical step: reasoning by absurdity or *Reduction ad Absurdum* (RAA). That is, if assuming a literal false (i.e. its default is true) leads to an inconsistency, then, in a 2-valued semantics, the literal must be true if that's consistent. SM does not do this because it requires every true literal to be supported by its rules. The solution proffered by rSM is to extend the notion of support to include reasoning by absurdity for this specific case, which reasoning is supported on the rules creating the odd loop. That is why the single rSM of a ← ~a is {a}.

It may be argued that SM employs odd loops as integrity constraints (ICs), but the problem remains that in program composition unforeseen odd loops may appear. rSM instead treats ICs specifically, by means of odd loops but involving for the purpose a reserved literal *'falsum'*, thereby separating the two issues, and so having it both ways, i.e. dealing with odd loops and ICs.

SM envisages default literals as assumptions that should be maximally true (the Closed World Assumption or CWA), on the proviso of stability. That is, that the conclusions following from the assumptions do not go against these. To the contrary, the whole model is confirmed by them, through the support of program rules. rSM takes this reasoning all the way, but relies on RAA to lend support to the model atoms introduced to resolve odd loops.

Whereas in the Well-Founded Semantics (WFS) the truth of literals, be they positive or default, may be interpreted as provability justified by a well-founded derivation, the lack of provability does not result in their falsity, because a third logical value, 'undefined', is available. In SM, though 2-valued, there is no general notion of provability defined, and one resorts to the interpretation of default negations as assumptions. The rSM view is that assumptions be revised, in a 2-valued way, if they would otherwise lead to self-inconsistency through odd loops.

That rSM resolves the inconsistencies of odd loops of SM (and note they are not contradictions, for there is no explicit negation) does not mean rSM should resolve contradictions. That is an orthogonal problem, and whose solutions can be added to different semantics, including rSM. Accordingly, in the "Extensions" section, we shall mention solutions to contradictions in Extended Logic Programs (ELPs) and Generalized Logic Programs (GLPs), inspired by the RAA approach. But these are optional, separate, add-ons.

The paper's remaining structure starts with a section on the definition of Revised Stable Models, justification, and examples; then another section presents two rSM semantics-preserving program transformations into NLP without odd loops; the next section contemplates properties of rSM and contrasts them with SM's; forthwith, comes a section on implementation; subsequently we describe the extension of rSM to explicit negation (ELPs) to produce Revised Answer Sets; the last section addresses conclusion and future work, as well as potential use. For lack of space, an Appendix includes Proofs of Theorems, and extensions to 'not's in heads (GLPs), and contradiction removal by belief revision [1].

Revised Stable Models

A Normal Logic Program (NLP) is a finite set of *rules* of the form H ←B1, B2, ..., Bn, not C1, not C2, …, not Cm (n, m ≥ 0) comprising positive literals H, Bi, and Cj, and default literals not Cj. Often we use '~' for 'not'.

Models are two-valued and represented as sets of the positive literals which hold in the model. The set inclusion and set difference mentioned below are with respect to these positive literals. Minimality and maximality too refer to this set inclusion.

Definition 1 (Gelfond-Lifschitz Γ operator [4]): Let P be a NLP and I a 2-valued interpretation. The GL-transformation of P modulo I is the program P/I, obtained from P by performing the following operations:

- remove from P all rules which contain a default literal not A such that A ∈ I
- remove from the remaining rules all default literals

Since P/I is a definite program, it has a unique least model J: Define Γ(I) = J. Stable Models are the fixpoints of Γ.

Definition 2 (Revised Stable Models and Semantics): M is a Revised Stable Model of a NLP P, where we let RAA(M) ≡ M – Γ(M), iff

- M is a minimal (classical) model
- RAA(M) is minimal not counting empty RAAs
- $\exists\, \omega \geq 2\ \Gamma^{\omega}(M) \supseteq RAA(M)$

The Revised Stable Models semantics is the intersection of its models, just as the Stable Model semantics is. Next we explain the function and justification of each condition above.

M is a minimal (classical) model – A classical model of a NLP is one that satisfies all its rules, where default negation is seen as classical negation. Satisfaction means that for every rule body true in the model its head too must be true in the model. Minimality of classical models is required to ensure maximal supportedeness (i.e., any true head is supported on a true body), compatible with model existence.

Stable Models are supported minimal classical models, and we wish to keep them in rSM as a special case. This condition includes them. In fact SM are the special case when there are no odd loops over negation. However, not all rSM are SM since odd loops of an atom over negation obtaining in a model are allowed in rSM to be resolved for the positive value of the atom. Nevertheless, this is to be achieved in a minimal way, i.e. resolving a minimal set of such atoms so that no odd loops obtain anymore. And justified through its logical "support" on a specific Reductio Ad Absurdum (RAA) application to that effect.

Example 2: Let P be {a ← ~a ; b ← ~a}. The only candidate minimal model is {a}, since {} and {b} are not models in the classical sense and {a, b} is not minimal. The need for Reductio ad Absurdum reasoning comes from the requirement to resolve odd loops over negation – an issue not dealt with in the traditional Stable Model semantics. In P, Γ({a}) = {} ⊆ {a}. Only with model {a} do we have set inclusion. The truth-value of 'a' is supported by a specific RAA on '~a' just in case it leads inexorably to 'a'. The first rule

forces ‘a’ to be in any possible model under the new semantics. I.e., assuming ‘a’ is not in a model, i.e. ‘~a’ is true, then the first rule insists that ‘a’ is in the model – an inconsistency. But if ‘~a’ cannot be true, and since the semantics is two-valued, then it must be false, and therefore ‘a’ must be true. So, the only model of this program must be {a}, since {b} is not a model, and {a, b} is not a minimal model with respect to the model {a}.

The third condition, explained below, aims at testing the inexorability of a default literal implying its positive counterpart, given the context of the remaining default literals being assumed in the model. The $\Gamma(M) \subseteq M$ property allows atoms to be minimally added to M over and above those of SMs, since these are defined as $\Gamma(SM) = SM$. The candidate additional atoms are specified in the next condition, namely those in $RAA(M)=M - \Gamma(M)$.

RAA(M) is minimal not counting empty RAAs – Indeed, we want the models which are most supported on themselves – so the $RAA(M) = M - \Gamma(M)$ should be minimal with respect to each other. Keep in mind that M must be a minimal model in a classical sense, which necessarily guarantees that $M \supseteq \Gamma(M)$. The maximum $\Gamma(M)$ can be is M. In the case of Stable Models $SM = \Gamma(SM)$, so $\Gamma(SM)$ is maximum, and RAA(SM) empty. The “not counting empty RAAs” part of this second condition serves the purpose of assuring that any Stable Model is also accepted as a Revised Stable Model as far as this condition is concerned. The minimality beyond empty RAAs lets us consider as rSM more minimal models than just the Stable Models.

Indeed, it is too strong to impose just RAA(M) minimality, among all their kind, that is including empty RAAs, because otherwise desirable rSMs may be thrown out when some SM exists, as its RAA(SM) is empty (cf. Example 4 below).

Example 3: a ← ~a b ← ~a c ← ~b

M1={a, c} is a minimal model. $\Gamma(M1)=\{c\}$ is maximal. $RAA(M1)=\{a\}$ is minimal. M2={a, b} is a minimal model. $\Gamma(M2)=\{\}$ which is NOT maximal. $RAA(M2)=\{a, b\}$ is NOT minimal (even not counting {}).

When a NLP has SMs, each verifies $SM=\Gamma(SM)$ and $RAA(SM)=\{\}$, which is minimal. The minimality condition on RAA(M) ensures that every SM is a rSM in regard to this condition. Furthermore, a NLP with SMs may have other rSMs which are not SMs, as shown next.

Example 4: c ← a, ~c a ← ~b b ← ~a

M1={b} is a minimal model. $\Gamma(M1)=\{b\}$ is maximal. $RAA(M1)=\{\}$ is minimal. M2={a, c} is a minimal model. $\Gamma(M2)=\{a\}$ is maximal. $RAA(M2)=\{c\}$ is NOT minimal. We have as rSMs {b}, its unique SM, and {a, c}, which has maximal supportedness of its literals. Although RAA(M2) is not minimal, precisely because there exists a SM, it is minimal not counting the empty $RAA(M1) = \{\}$..

Example 5: a ← ~b b ← ~a, c c ← a

There is a single SM1={a, c}, Γ(SM1)={a, c}, RAA(SM1)={}. There are two rSM: rSM1=SM1={a, c} and rSM2={b}. rSM2 respects all three rSM conditions; note how the not counting {} proviso is essential for rSM2 because of SM1's existence, given that Γ(rSM2)={}, RAA(rSM2)={b}. $\Gamma(\Gamma(\text{rSM2})) = \Gamma(\{\}) = \{a, b, c\} \supseteq \text{RAA(rSM2)}$.

Example 6: a ← ~a, ~b d ← ~a b ← d, ~b

M1={a}, Γ(M1)={}, RAA{a}, and M2={b, d}, Γ(M2)={d}, RAA(M2)={b} are both rSMs.

Conceivably, one might think a legitimate non-minimal RAA(M) could be obtained by the union of maximal RAA literals from disjoint models of disjoint subprograms, for some M that obeys the other conditions. And, in that case, that union might not be minimal. Such RAA(M) might nevertheless be desirable, and so the minimality condition could be considered too strict. Instead of allowing them as rSMs, we shall call them the *Combination Revised Stable Models* (CrSMs). CrSMs are then an extension to standard rSMs. The possible acceptance of non-minimal RAA(M) can then be justified by the CrSMs obtained from the disjoint subprograms (cf. motivating Example 7 below).

Example 7: a ← ~b b ← ~a c ← a, ~c
x ← ~y y ← ~x z ← x, ~z

M1={b, y}, M2={a, c, y}, M3={b, x, z}, are its rSMs.
Γ(M1)={b, y}, Γ(M2)={a, y}, Γ(M3)={b, x}.
RAA(M1)={}, RAA(M2)={c}, RAA(M3)={z}.

If we take M4={a, c, x, z}, we can see that Γ(M4)={a, x}, RAA(M4)={c, z}, and $\Gamma(\Gamma(\{a,c,x,z\})) = \Gamma(\{a,x\}) = \{a,c,x,z\} \supseteq \text{RAA(M4)} = \{c,z\}$. M4 respects all three definition conditions except for the RAA(M4) minimality one, because it is greater than RAA(M2), and also than RAA(M3). M4 is therefore not a rSM. However, it could be interesting to consider it as a possible candidate model of the program. Such CrSMs provide an extension to rSMs, of which M4 is an example.

The formal definition of Combination Revised Stable Models can be found in the Extensions section of the paper. Note CrSMs are not really needed for top-down querying, since they exist as a result of disjoint subprograms with separate rSMs, and any goal will appeal to just one such subprogram. Moreover, a rSM can be extended to a CrSM if desired.

$\exists\, \omega \geq 2\ \Gamma^{\omega}(M) \supseteq \text{RAA}(M)$ – For the sake of the explanation let us first start with a more verbose, but also more intuitive version of this condition:

$\exists\, \omega \geq 0\ \Gamma^{\omega}(\Gamma(\text{M-RAA(M)})) \supseteq \text{RAA(M)}$ where $\Gamma^{0}(X) = X$ for any X

Since RAA(M)=M–Γ(M), it can be understood as the subset of literals of M whose defaults are self-inconsistent, given the rule-supported literals in Γ(M), the SM part of M. The RAA(M) are not obtainable by Γ(M). The condition states that successively applying the Γ operator to M-RAA(M), i.e. to Γ(M), which is the "non-inconsistent" part of the model or rule-supported context of M, we will get a set of literals which, after ω

iterations of Γ if needed, will get us the RAA(M). RAA(M) is thus verified as the set of self-inconsistent literals, whose defaults RAA-support their positive counterpart.

This is intuitively correct: assuming the self-inconsistent literals as false they appear later as true consequences. We can simplify this expression to $\exists\ \omega\geq 0\ \Gamma^{\omega}(\Gamma(\Gamma(M))) \supseteq RAA(M)$. And then to $\exists\ \omega\geq 2\ \Gamma^{\omega}(M) \supseteq RAA(M)$, to obtain the original one. Of course, all SMs comply with this condition because in their case RAA(SM)={}. So, for SMs all our three rSM conditions reduce back to their usual definition of Γ(SM)=SM.
The approach to the third condition is inspired by the use of Γ and Γ^2, in one definition of the Well-Founded Semantics (WFS), to determine the undefined literals. We want to test that the atoms in RAA(M) introduced to resolve odd loops, actually lead to themselves through repeated (at least 2) applications of Γ, noting that Γ^2 is the consequences operator appropriate for odd loop detection, as seen in the WFS, whereas Γ is appropriate for even loop SM stability. Because odd loops can have an arbitrary length, repeated applications are required. Because even loops are stable in just one application of Γ, they do not need iteration, which is the case with SMs.

The non-monotonic character of Γ, when coupled with the existence of odd loops, may produce temporary spurious elements not in M in the second application of Γ in Γ^2, and hence the use of set inclusion in the condition. No matter, because the test is just to detect that introduced atoms additional to Γ(M) actually are supported by RAA. On the other hand, such spurious atoms do not persist, for they disappear in the next application of Γ.

Because odd loops over negation can have arbitrary length, we need the number of iterations of Γ to be unlimited a priori.

Example 8: a ← ~b t ← a, b k ← ~t
b ← ~a i ← ~k

M1={a,k}, Γ(M1)= {a,k}, RAA(M1)={}, Γ(M1)⊇RAA(M1). M1 is a rSM.
M2={b,k}, Γ(M2)= {b,k}, RAA(M2)={}, Γ(M2)⊇RAA(M2). M2 is a rSM.
M3={a,t,i}, Γ(M3)= {a,i}, RAA(M3)={t}, $\sim\exists\ \omega\geq 2\ \Gamma^{\omega}(M3) \supseteq RAA(M3)$. M3 is not a rSM.
M4={b,t,i}, Γ(M4)= {b,i}, RAA(M4)={t}, $\sim\exists\ \omega\geq 2\ \Gamma^{\omega}(M4) \supseteq RAA(M4)$. M4 is not a rSM.

Although Γ(M3) and Γ(M4) are maximal, from neither is 't' obtainable by iterations of Γ. Simply because '~t', implicit in both, is not conducive to 't' through Γ. This is the purpose of the third condition. The attempt to introduce 't' into RAA(M) fails because RAA cannot be employed to justify 't'.

Example 9: a ← ~b b ← ~c c ← ~a

M1={a,b}, Γ(M1)={b}, RAA(M1)={a}, Γ^2(M1) = {b,c}, Γ^3(M1) = {c}, Γ^4(M1)={a,c} ⊇ RAA(M1). The remaining Revised Stable Models, {a,c} and {b,c}, are similar to this one, by symmetry.

It took us 4 iterations of Γ to get a superset of RAA(M) in a program with an odd loop of length 3. In general, a NLP with odds loops of length N will require ω = N+1 iterations

of the Γ operator. Let us see why this is so. First we need to obtain the supported subset of M, which is Γ(M). The RAA(M) set is precisely the subset of M that does not intersect Γ(M), so under Γ(M) all literals in RAA(M) have truth-value 'false'. Now we start iterating the Γ operator over Γ(M). Since the odd loop has length N, we need N iterations of Γ to finally make arise the set RAA(M). Hence we need the first iteration of Γ to get Γ(M) and then N iterations over Γ(M) to get RAA(M) leading us to ω = N+1. In general, if the odd loop lengths are decomposed into the primes {N1,…,Nm}, then the required number of iterations, besides the initial one, is the product of all the Ni.

It may be argued that SM employs odd loops as integrity constraints (ICs), but the problem remains that in program composition unforeseen odd loops may appear. rSM treats ICs specifically, by means of odd loops involving a reserved literal *'falsum'*, whose truth is disallowed in every model, thereby separating the two issues, and so having it both ways.

Definition 3 (Integrity Constraints): Incorporating Integrity Constraints (ICs) in a NLP under the Revised Stable Models semantics consists in adding a rule of the form

falsum ← an_IC, ~falsum

for each IC. 'falsum' is a reserved atom, required to be false in all models. The 'an_IC' in the rule stands for a conjunction of literals, which must not be true, that form the IC.

From the odd loop introduced this way it results that, whenever 'an_IC' is true, 'falsum' must be in the model, a contradiction. Consequently only models where 'an_IC' is false are allowed. Whereas in SM odd loops are used to express ICs, in rSM they are too, but using the reserved 'falsum' predicate.

Transformations into normal programs

Two program transformations are provided next, such that the SMs of the transformed program correspond exactly to the rSMs of the original one.

Definition 4 (The RAA transformation)

Consider M is some rSM of P. For each literal A in M – Γ(M) add to P the set O of rules of the form

A ← not_M

to obtain program Podd, where not_M stands for the conjunction of default negations of each element NOT in M. The rules in O add to P, depending on context not_M, the atoms A exactly required to resolve odd loops which would otherwise prevent P from having a SM in that context. Since one can add to Podd the O rules for every context not_M, the Stable Models of the transformed program Podd = P U O are the rSM of Podd. Moreover, one can add to Podd all such rules for all its models M.

Example 10: Let P be a ← b, ~a b ← ~c c ← ~b

M1={c}, O1={}, M2={a,b}, O2={a ← ~c}. O=O1 U O2, and the SMs of P U O are the rSMs of P.

Theorem 1 – Correctness of RAA transformation: The RAA transformation is correct.

Proof (sketch): (cf Appendix A – Theorems' Proofs).

Consequently, if one knew all alternative RAA sets for P, one could implement the rSM semantics via this transformation plus resorting to an implementation of SMs. But detecting the RAAs is not immediate. Hence this transformation is more of a theoretical interest, say for proving properties, namely that rSMs always exist (as there are no odd loops in the transformed program) and that they are cumulative (for the same reason) , as we shall do below.

Definition 5 (The EVEN transformation): Next we provide a program transformation, EVEN: NLP → NLP, for a normal program P, so that M is a rSM of P iff M is a SM of the transformed program Pf, in respect to the intersection of the languages of P and Pf, and which maximizes new literals of the form *L_f*. Pf is the NLP resulting from the application of the EVEN transformation to the NLP P, where Pf = EVEN(P) iff:

$$EVEN(P) = Tf(P) \cup Ct\text{-}tf(P)$$

and Tf(P) is the result of substituting, in each rule of P, each default literal ~*L* in P by a new positive literal *L_f* not yet existent in P, and Ct-tf(P) is the set of pairs of new rules of the form (creating even loops):

$$L \leftarrow \sim L\_f \qquad L\_f \leftarrow \sim L$$

for each literal *L* with rules in P. Literals without rules in P are not translated into Ct-tf(P) pairs. Instead, they are translated into *L_f* ←, i.e. their correspondent negative literals are always true. These are the default literals necessarily true by CWA in all models.

The basic ideas of the transformation are:

1. No odd loops exist in Pf.
2. Literals can have true or false values, by means of the newly introduced even loops between *L* and ~ *L_f* , but default literals without rules in P become true *L_f* literals.
3. Odd loops in P prevent assuming ~*L_f* . Eg. c ← ~c translates into c ← c_f which, together with the even loop c ← ~c_f c_f ← ~c , prevents assuming c_f , which would be self-defeating, I.e. assuming 'c_f' one has 'c' by implication, but then 'c_f' is not supported by its only rule, c_f ← ~c, and so cannot belong to the SM.
4. Maximizing the *L_f* literals guarantees the CWA.

Example 11:

a ← ~b	EVEN(P)	a ←b_f	a ← ~a_f	b ← ~b_f	c ← ~c_f
b ← ~a	========>	b ← a_f	a_f ← ~a	b_f ← ~b	c_f ← ~c
c ← a, ~c, ~d		c ← a, c_f, d_f		d_f ←	

Theorem 2 – Correctness of the EVEN transformation: The EVEN transformation is correct.

Proof (sketch): (cf Appendix A – Theorems' Proofs).

Example 12: Take now the program a ← ~b , that translates into a ← b_f plus the rule b_f ← , and the even loop for 'a' and 'a_f' (not shown). Note that no even loop between 'b' and 'b_f' is required. If introduced, instead of b_f ←, then one would have to maximize on *L_f* literals (for achieving the CWA) so the even loop would be resolved in favour of 'b_f'.

Properties

Theorem 3 – Existence: Every NLP has at least one Revised Stable Model.

Proof (sketch): (cf Appendix A – Theorems' Proofs).

Theorem 4 – Stable Models extension: Every Stable Model of an NLP is also a Revised Stable Model of it.

Proof (sketch): (cf Appendix A – Theorems' Proofs).

SM does not deal with Odd Loops Over Negation, except to prohibit them, and that unfortunately ensures it does not enjoy desired properties such as *Relevance*. For example, take a program such as:

c ← a, ~c a ← ~b b ← ~a

Although it has an SM={b} it is non-relevant, e.g. in order to find out the truth-value of the literal 'a' we cannot just to look below the rule dependency call graph for 'a', but need also to look at all other rules that depend on 'a', namely the first rule for 'c'. This rule in effect prohibits any SM containing 'a' because of the odd loop in 'c' arising when 'a' is true, i.e. 'c ← ~c'. Hence, as the example illustrates, no top down call graph based query method can exist for SM, because the truth of a literal potentially depends on all of a program's rules.

Relevance is the property that makes it possible to implement a top-down call-directed query-derivation proof-procedure – a highly desirable feature if one wants an efficient theorem-proving system that does not need to compute a whole model to answer a query. The non-relevance of Stable Models, however, is caused exclusively by the presence of odd loops over default negation, as these are the ones that may render unacceptable a partial model compatible with the call-graph below a literal. Even loops can accommodate the partial solution by veering one direction or the other.

rSMs, by resolving odd loops in favour of their heads, effectively preventing their constraining hold on literals permitting the loop, enjoys relevance, and is thus potentially amenable to top-down call-graph based query methods (and we shall touch upon one in the implementation section). These methods are designed to try and identify whether a

query literal belongs to some rSM, and to produce the partial rSM supporting a positive answer, which can be potentially extended, because of relevance, to a full rSM.

Theorem 5 – Relevance: The Revised Stable Models semantics is Relevant.

Proof (sketch): (cf Appendix A – Theorems' Proofs).

Theorem 6 – Cumulativity: The Revised Stable Models semantics is Cumulative.

Proof (sketch): (cf Appendix A – Theorems' Proofs).

Example 13: a ← ~b b ← ~a, c c ← a

There is a single SM1={a, c}. If 'c ←' is added, then there is an additional SM2={b, c}, and cumulativity for SM fails because 'a' no longer belongs to the intersection of SMs. There are two rSM: rSM1=SM1={a, c} and rSM2={b}; 'c ←' cannot be added.

Cumulativity pertains to the intersection of models, which defines the SM and the rSM semantics. But seldom is this intersection used in practice, and SM implementations are focused on computing the set of models.

Another but similar notion of cumulativity pertains to storing lemmas as a proof develops, giving rise to the techniques of memoizing and tabling in some Prolog and WFS systems. This is a nice property which ensures one can use old computation results from previous steps in a query-oriented derivation to speed up the computation of the rest of the query by avoiding redundant computations. This cumulativity presupposes a top-down call-graph oriented query derivation method exists, which is the case for rSM because it affords relevance, but not for SM.

For this second type of cumulativity relevance is again essential because it guarantees that the truth of a literal depends only on the derivational context provided by the partial rSM supporting the derivation, namely the default literals which are true in it. Consequently, if a positive literal A is found true in context not_C standing for the conjunction of default literals true in it, then a rule may be added to that effect, namely A ← not_C or, better still for efficiency, entered into a table.

Implementation

Since the Revised Stable Models semantics is Relevant (see proof sketch of Theorem 5 above) it is possible to have a top-down call-directed query-derivation proof-procedure that implements it.

One such procedure to find out (querying) if a literal A belongs to a Revised Stable Model M of a NLP P can be viewed as finding a derivational context, i.e. the truth-value, of the required default literals in the Herbrand base of P under that model M, such that A follows, plus the required literals true by RAA in that derivation. The first requirement is

simply finding an abductive solution, considering all default negated literals as abducibles, that forms a default literal context which supports A.

An already implemented system, tested, and with proven desirable properties – such as soundness and completeness –that can be adapted to provide both requirements is ABDUAL [3]. ABDUAL defines and implements abduction over the Well-Founded Semantics for extended logic programs (i.e.. normal programs plus explicit negation) with integrity constraints (ICs), by means of a query driven procedure. This proof procedure is also defined for computing Generalized Stable Models (GSM), i.e. NLPs plus ICs, by considering as abducibles all default literals, and imposing that each one must be abduced either true or false, in order to produce a 2-valued model.

This is so because the ABDUAL procedure also accounts for the Generalized Stable Models (GSM) semantics and can evaluate abductive queries over GSM programs. ABDUAL needs to be adapted in two ways to compute partial rSMs in response to a query. First, the 2-valued ICs must be relaxed so that only default literals visited by a relevant query driven derivation are imposed 2-valuedness. Literals not visited remain unspecified, because the partial rSM obtained can always be extended to all default literals because of relevance. Second, ABDUAL must be adapted to detect literals involved in an odd loop with themselves, so that RAA can then be applied, thereby including such literals in the (consistent) set of abduced ones. The reserved 'falsum' literal is the exception to this, so that ICs can be implemented as explained before, including the ICs imposing 2-valuedness on rSMs.

The publicly available interpreter for ABDUAL for XSB-Prolog is modifiable to comply with these requirements. A more efficient solution involves adapting XSB-Prolog to enforce the two requirements at a lower code level. (cf. [3] for the details). These alterations correspond, in a nutshell, to small changes in the ABDUAL meta-interpreter.

The EVEN transformation given can readily be used to implement rSM by resorting to some implementation of SM, such as the SMODELS or DLV systems. In that case full models are obtained and no query relevance can be enacted, of course. *L_f* are maximize by resorting to commands in these systems.

Extensions

Combination Revised Stable Models

When a Normal Logic Program can be divided into two disjoint subprograms – no atom in one subprogram occurs in the other subprogram – it can make sense to consider as a model of the whole combined program the union of the disjoint Revised Stable Models of each of the subprograms. Combination Revised Stable Models (CrSMs) meet this requirement.

Definition 6 (Combination Revised Stable Models):
Let P be a NLP and P1 and P2 subprograms (subsets of rules) of P such that their sets of atoms are disjoint. Let M1 be a rSM of P1 and M2 one of P2. If there is a minimal model

CRM of P such that RAA(CRMM) = RAA(M1) U RAA(M2), and $\exists\ \omega \geq 2\ \Gamma^{\omega}(CRM) \supseteq$ RAA(CRM), then M is a Combination Revised Stable Model of P.
Now allow CrSMs to be used also, iteratively, for the "construction" of other CrSMs. in the same way. The class of CrSMs is then forthwith defined.

Example 14: Let is consider P as a ← ~b b ← ~a, c c ← a
x ← ~y y ← ~x, z z ← x

Its rSM are M1={a,c,x,z}, Γ(M1) = {a,c,x,z}, RAA(M1) = {} (M1 is the only SM)
M2 = {a,c,y}, Γ(M2) = {a,c}, RAA(M2) = {y}
M3 = {b,x,z}, Γ(M3) = {x,z}, RAA(M3) = {b}

Its CrSMs include all the rSMs and also M4, the only other CrSM:
M4 = {b,y}, Γ(M4) = {}, RAA(M4) = {b,y} = RAA(M2) U RAA(M3)

M4 is a minimal model, and Γ(Γ(M4)) = Γ(Γ({b,y})) = Γ({}) = {b,y} ⊇ RAA(M4) = {b,y}; so $\exists\ \omega \geq 2\ \Gamma^{\omega}(M4) \supseteq$ RAA(M4); ω = 2.

Extended Logic Programs

Extended LPs (ELPs) introduce explicit negation into the syntax of NLPs. Each positive atom may be preceded by '-', standing for explicit negation, whether in heads, bodies, or arguments of 'nots'. Positive atoms and their explicit negations are collectively dubbed "objective literals". For ELPs, SM semantics is replaced by Answer-Set semantics (AS) [6], coinciding with SM on NLPs. AS employs the same stability condition on the basis of the Γ operator as in SM, treating all objective literals as positive, and default literals as negative.

Furthermore, its models (the Answer-Sets) must be non-contradictory, in the sense of not containing a positive atom and its explicit negation, otherwise a single model exists, and it is comprised of all objective literals, that is, from a contradiction everything follows. Note that Answer-Sets (ASs) need not contain an atom or its explicit negation, that is, explicit negation does not comply with the Excluded Middle principle, like classical negation does. Furthermore, it is a property of AS that, for any 'L' of the form 'A' or '-A' where 'A' is a positive atom, if '-L' is true then 'not L' is true as well (Coherence).

Definition 7 (Extension to Answer-Sets – Revised Answer-Sets (rAS)): rSM can be naturally applied to ELPs, by extending AS in a similar way as for SM, thereby obtaining rAS (Revised Answer Sets), which does away with odd loops but not the contradictions brought about by explicit negation. The same definition conditions apply as for rSM, plus the same proviso on contradictory models as in AS (if a contradiction exists there is a single rAS model comprised of the whole Herbrand base). The consequences are therefore similar too.

Example 15: Under rSM, let P be a ← ~b b ← ~c c ← ~a

The rSMs of P are {a, b}, {b, c}, and {a, c}. If we consider instead the rAS setting and a slightly different version of the program with explicit negation (replacing 'c' with '-a'), under rAS let P' be

a ← ~b b ← ~ -a -a ← ~a

The rASs of P' are {a, b} and {b,-a}; the correspondent {a,-a} from P is rejected under rAS because it is contradictory.

Example 16: Under rSM, let P be a ← ~a b ← ~b

The rSM of P is just {a, b}. If we consider instead the rAS setting and a slightly different version of the program with explicit negation (replacing 'b' with '-a'), under rAS let P' be

a ← ~a -a ← ~-a

there will be no non-contradictory rASs since the only possible correspondent candidate is {a,-a}.

Other extensions are described in the Appendix B.

Conclusions and future work

Having defined a new 2-valued semantics for normal logic programs, and having proposed more general semantics for several language extensions, much remains to be explored, in the way of properties, comparisons, implementations, and applications, contrasting its use to other semantics employed heretofore for knowledge representation and reasoning.

The fact that rSM includes SMs and the virtue that it always exists and admits top-down querying is a novelty that may make us look anew at the use of 2-valued semantics of normal programs for knowledge representation and reasoning [1].

Worth exploring is the integration of rSM with abduction, whose nature begs for relevance, and seamlessly coupling 3-valued WFS (and extensions) implementation such as XSB-Prolog, with 2-valued rSM implementations, such as the modified ABDUAL or the EVEN transformation, so as to combine virtues of both bringing closer together the 2- and 3-valued logic programming communities.

Another avenue is in using rSM and its extensions, in contrast to SM based ones, as an alternative base semantics for updatable and self-evolving programs [5, 2] so that model inexistence after an update may be prevented in a variety of cases. This may be of significance to semantic web reasoning, a context in which programs may be being updated and combined dynamically from a variety of sources.
rSM implementation, in contrast to SM's ones, because of its relevance property can avoid the need to compute whole models and all models, and hence the need for groundness and the difficulties it begets for problem representation. Naturally it raises problems of constructive negation, but these are not specific to or begotten by it. Because it can do without groundness, meta-interpreters become a usable tool and enlarge the degree of freedom in problem solving.

In summary, rSM has to be put the test of becoming a usable and useful tool. And first of all by persuading researchers that it is worth using, and worth pursuing its challenges.

Acknowledgements

For comments on work leading to the paper: J. Alferes, F. Banti, Pierangelo Dell'Acqua, M. Gelfond, P. Hitzler, R. Kahle. For partial support: projects POCTI FLUX in Portugal, DAAD IQN "Rational Mobile Agents" with Dresden, EU FP6 NoE REWERSE, and, especially, sabbatical scholarships from FCT, Portugal, and Istituto di Studi Avanzati, U. Bologna, Italy to the first author, without which enough peace of mind would not have been had.

References

1. J. J. Alferes, L. M. Pereira, *Reasoning with Logic Programming*, LNAI 1111, Springer-Verlag, 1996.
2. J. J. Alferes, A. Brogi, J. A. Leite ,L. M. Pereira. *Evolving Logic Programs*. In S. Flesca et al. (eds.), Procs. 8th European Conf. on Logics in AI (JELIA'02), pp. 50-61, Spriger, LNCS 2424, 2002.
3. J. J. Alferes, L. M. Pereira, T. Swift, Abduction in Well-Founded Semantics and Generalized Stable Models via Tabled Dual Programs, *Theory and Practice of Logic Programming*, 4(4):383-428,July 2004.
4. M. Gelfond, V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski, K. A. Bowen (eds.), Fifth International Logic Programming Conference, pp. 1070-1080. MIT Press, 1988.
5. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, Dynamic Updates of Non-Monotonic Knowledge Bases, *The Journal of Logic Programming* 45(1-3):43-70, Sept/Oct 2000.
6. M. Gelfond, V. Lifschitz. Logic Programs with classical negation. In D.S.Warren, P.Szeredi (eds.), 7th International Logic Programming Conference, pp. 579-597. MIT Press, 1990.
7. J. Dix. A Classification Theory of Semantics of Normal Logic Programs: I. Strong Properties, II. Weak Properties, Fundamenta Informaticae XXII(3)*227—255, 257–288, 1995.
8. P. M. Dung, On the Relations between Stable and well-founded Semantics of Logic Programs, Theoretical Computer Science, 1992, Vol 105, pp 7 - 25, Elsevier Publishing B.V.

Appendixes

Appendix A – Theorems' Proofs

Theorem 1 – Correctness of RAA transformation: The RAA transformation is correct.

Proof (sketch): The RAA atoms A introduced by set of rules O, are so introduced just in case its corresponding context not_M is in force. Thus if there exists a SM model of Podd containing not_M it will be exactly one, M. The reason it exists is that all odd loops appearing in context not_M in P are resolved by definition, by including in M the corresponding RAA literals provided by the rules O. The truth of these literals justifies the non-inclusion in M of their corresponding defaults. Since there are no odd loops in context not_M, and all RAA literals are now supported, just like the others, M will be a SM of Podd. This being valid for any context not_M, the SMs of Podd coincide with the rSMs of P. QED

Theorem 2 – Correctness of the EVEN transformation: The EVEN transformation is correct.

Proof (sketch): The EVEN transformation retains all candidate model possibilities, by introducing an even loop between every literal with rules and its default negation, the latter now expressed as a positive literal. It is these independent even loops that, by themselves alone, allow for all oscillation combinations of such literals and their defaults. When they are joined to the remaining rules of the transformation, two issues have to be taken care of. One is that CWA be achieved by the simpler transformation for literals without rules. Defaults of literals with rules, are either made false by their positive counterparts, or else participate in even or odd loops. The even loops will allow them to oscillate. The odd loops constitute the second issue. By the nature of the transformation, an odd loop actually obtaining between a literal and its default gives rise to a choice on how the even loop between their translations is broken because *L_f* will imply *L* (cf example above). If *L* is true anyway by other rules, no harm is done; otherwise the RAA duly takes place. And this occurs only on a by need basis. Consequently, the SMs of the transformed program will correspond to the rSMs of the original program, once the vocabulary translation is reversed. Because of the CWA transformation rule, only those SMs maximizing the *L_f* are retained. QED

Theorem 3 – Existence: Every NLP has at least one Revised Stable Model.

Proof (sketch): We visit in turn the three defining conditions of rSM. Every NLP has at least one minimal classical model M, with its corresponding $\Gamma(M)$. And at least one such $\Gamma(M)$ will be maximal. If for some or all M $M=\Gamma(M)$ then $RAA(M)=\{\}$, and the third conditions is trivially satisfied. Otherwise, if $M=\Gamma(M)$ for no M then there are no SMs. And so there exists at least one odd loop over default negation justifying this, for even loops alone cannot prevent the existence of SMs, as even loops can always go one way or the other or both, as proven in [8]. Accordingly, there exists at least one non-empty RAA(M) which is the head of an odd loop. For all such elements in RAA(M) the third condition is satisfied, for it detects whether such heads are supported on their defaults, in the context of the remaining model. QED

Theorem 4 – Stable Models extension: Every Stable Model of an NLP is also a Revised Stable Model of it.

Proof (sketch): The three defining conditions of rSM are satisfied by all SMs. Every SM is a minimal classical model. No SM is a subset of another, hence all are maximal with respect to one another. No SM is a subset of a non-SM rSM, and hence all are maximal with respect to them, the reason being as follows: for any non-SM rSM its RAA(M) is non-empty, corresponding to least one resolved odd loop; but in any SM this loop is broken, otherwise the SM would not exist; consequently, no SM is compatible with a non-SM rSM. For SMs RAA(M) is empty, and so for them the third condition is trivially true. QED

Theorem 5 – Relevance: The Revised Stable Models semantics is Relevant.

Proof (sketch): The semantically equivalent and correct RAA program transformation above does away with odd loops over negation. This is so because any context 'not_M' where such a loop occurs, for say head 'A', gives rise to a new rule 'A ← not_M' which makes 'A' true in that context, thereby sidetracking the loop. Consequently, only even loops above a query might remain active. But these, as we have remarked, can adapt to the constraints imposed on them by the query result. QED

Theorem 6 – Cumulativity: The Revised Stable Models semantics is Cumulative.

Proof (sketch): First recall some formal property definitions that will be used, where T |= A signifies that A belongs to all SMs of T, i.e. their intersection.

> Monotonicity: T |= A => T U {B} |= A
> Weak or Cautious Monotonicity: T |= A and T |= B => T U {B} |= A
> Cumulativity: T |= B => (T |= A <=> T U {B} |= A)
> Cut: T |= B and T U {B} |= A => T |= A
> Cautious Monotonicity plus Cut equals Cumulativity. SM semantics enjoy Cut. [7].

As the SM semantics enjoys the Cut, and since the SMs of program Pood, in the RAA program transformation, are equivalent to the rSMs of P, it suffices to prove that the SMs of Pood also enjoy the Cautious Monotonicity, in order to prove the Cumulativity of rSM. Let M be a Stable Model of Pood and A and B any elements of M. Pood has no odd loops over negation, as argued in the proof about Relevance. But only odd loops can prevent the appearance of models, or make new models appear because they are no longer prevented. Consequently, adding literals in the intersection is not going to change any of the existing models, or add or subtract to the set of models. Thus their intersection remains the same and cumulativity is warranted. QED

Appendix B – Other Extensions

Revision Revised Answer Sets (rrAS)

An open issue is how to apply RAA to revise contradictions based on default assumptions, not just removing odd loops, defining then what might be called rrAS (Revision Revised AS). Thus instead of "exploding" a contradictory model into the Herbrand base, one would like to minimally revise default assumptions so that no contradiction appears in a model. Here is the definition, which needs only enhance the first condition of rSM:

Definition 7 (Revision Revised Answer Sets): M is a Revision Revised Answer Set of an Extended Logic Program P iff, where M= Γ(M) U RAA(M) RAA(M)=RAA1 U RAA2:

- M is a minimal (classical) model, with respect to objective literals, where no pair <L, -L> is allowed)
- Γ(M) is maximal, or RAA(M) is minimal not counting any RAA(M)={}, within their own kind
- ∃ RAA1, RAA2, L s.t. ∃ ω≥2 Γ^{ω}(RAA1) ⊇ RAA1 and ∃ ω≥2 Γ^{ω}(M-RAA2) ⊇ {L, -L}

Example 16: Let P be {-a ← ~b; a ← ~b; c ← ~c}. The only rrAS is {b,c}. How is 'b' supported? According to RAA, in general, when a set of assumptions leads to contradiction, they should be revised. In our case, the assumptions are default literals, and the revision is 2-valued. Since there is only one assumption in this example, '~b', it is revised to 'b'. RAA(M)={b,c}, RAA1={c}, RAA2={b}.

In our case, one wants minimal revisions since defaults are to be maximized. Sure, one revision may lead to a new contradiction, as in the case above if 'b' itself implied a contradiction. In that case there would be no model, a definite possibility when one is dealing with hard contradictions. But our definition already foresees that no contradictions are allowed, whatever revisions are in force. How do we know which literals were revised? Well those in RAA(M): they are the result of odd loop inconsistency resolution, or non Γ-supported revision atoms, or both. But these must be there by need, that is, they actually are necessary for some contradiction avoidance.

Generalized Logic Programs, their Stable Models semantics (GLP) and their Revised Stable Models semantics (rGLP)

Generalized LPs (GLPs) introduce default negated heads into the syntax of NLPs. For GLPs, SM semantics is replaced by GLP, coinciding with SM on NLPs [5]. It will be convenient to syntactically represent generalized logic programs as propositional Horn theories. In particular, we will represent default negation 'not A' as a standard propositional variable (atom). Suppose that K is an arbitrary set of propositional variables whose names do not begin with a 'not'. By the propositional language LK generated by the set K we mean the language L whose set of propositional variables consists of: {A : A ∈K} U {not A : A ∈ K}. Atoms A ∈ K, are called objective atoms

while the atoms 'not A' are called default atoms. From the definition it follows that the two sets are disjoint.

By a generalized logic program P in the language LK we mean a finite or infinite set of propositional Horn clauses of the form $L \leftarrow L1, \ldots, Ln$ where L and Li are atoms from LK. If all the atoms L appearing in heads of clauses of P are objective atoms, then we say that the logic program P is normal. Consequently, from a syntactic standpoint, a logic program is simply viewed as a propositional Horn theory. However, its semantics significantly differs from the semantics of classical propositional theories and is determined by the class of stable models defined below.

By a (2-valued) interpretation M of LK we mean any set of atoms from LK that satisfies the condition that, for any A in K, precisely one of the atoms A or not A belongs to M . Given an interpretation M we define:

$$M+ = \{A \in K : A \in M\} \qquad M- = \{\text{not } A : \text{not } A \in M\} = \{\text{not } A : A \notin M\}.$$

By a (2-valued) model $M= M+ \cup M-$ of a generalized logic program P we mean a (2-valued) interpretation of P that satisfies all of its clauses. A program is called consistent if it has a model. A model M is considered smaller than a model N if the set of objective atoms of M is properly contained in the set of objective atoms of N. A model of P is called minimal if there is no smaller model of P. A model of P is called least if it is the smallest model of P. It is well-known that every consistent program P has the least model $M = \{A : A \text{ is an atom and } P \models A\}$.

Definition 8 (Stable models of generalized logic programs): We say that a (2-valued) interpretation M of LK is a stable model of a generalized logic program P if M is the least model of the Horn theory $P \cup M-$:

$$M = \text{Least}(P \cup M-) \text{ or, equivalently, if } M = \{A : A \text{ is an atom and } P \cup M- \models A\}$$

.

Revised GLP semantics (rGLP)

Definition 9 (Revised Generalized Logic Program models): M is a Revised SM model of a GLP program P, where we let $RAA(M) \equiv M - \text{Least}(P \cup M-)$, iff :

- $M = M+ \cup M-$ is a minimal (classical) model, with respect to objective literals (no inconsistent pair <L, not L> allowed)
- Least(P U M-) is maximal, or RAA(M) is minimal not counting any RAA(M)={}, within their own kind
- [Least(M - RAA(M) U not RAA(M)] $\supseteq$ RAA(M), where 'not RAA(M)' stands for the set of negated elements in RAA(M)

These conditions are just adaptations of the rSM conditions for the GLP syntax and semantics. The third condition in particular states that the elements of RAA(M), those essential to resolve odd loops, must be obtainable through a Least operation on the set

resulting from deleting them from M and adding instead their default negations, which now the syntax allows. No matter that the resulting set is inconsistent, the test simply aims at checking that the RAA(M) atoms are indeed supported on their negations.