Revised Stable Models – a Semantics for Logic Programs

Luís Moniz Pereira and **Alexandre Miguel Pinto**
{lmp|amp}@di.fct.unl.pt
Centro de Inteligência Artificial – CENTRIA
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

Abstract

This paper introduces an original 2-valued semantics for Normal Logic Programs (NLP), which conservatively extends the Stable Model semantics (SM) to all normal programs. The distinction consists in the revision of one feature of SM, namely its treatment of odd loops, and of infinitely long support chains, over default negation. This single revised aspect, addressed by means of a *Reductio ad Absurdum* approach, affords a number of fruitful consequences, namely regarding existence, relevance and top-down querying, cumulativity, and implementation.
The paper motivates and defines the Revised Stable Models semantics (rSM), justifying and exemplifying it. Properties of rSM are given and contrasted with those of SM. Furthermore, these results apply to SM whenever odd loops and infinitely long chains over negation are absent, thereby establishing significant, not previously known, properties of SM. Conclusions, further work, terminate the paper.

Keywords Logic program semantics, Stable Models, *Reductio ad absurdum*

Introduction

The paper introduces a new 2-valued semantics for Normal Logic Programs (NLP), called Revised Stable Models semantics (rSM), cogent in the properties it enjoys. Its name intends to draw attention to being inspired by, and actually revising, Stable Model semantics (SM) [2]. Indeed SMs are just particular rSM models, and the definition of the SM is a specific instance or specialization of the rSM one. But its name also draws attention to that the definitional distinction between the two consists in the *revision* of one feature of SM, namely its treatment of odd loops over negation, as well as of infinite support chains over negation. Finally, this single revised aspect is addressed by means of a *Reductio ad Absurdum* approach, a form of belief *revision*, and affords a number of fruitful consequences, not shared by SM, the present 'de facto' standard for 2-valued semantics for NLP.

For one, rSM are guaranteed to *exist* for all NLP. The concrete examples below show that odd loops may be required to model knowledge. Moreover, this guarantee is crucial in program composition (say from knowledge originating in divers sources) so that the result has a semantics. It is also important to warrant the existence of semantics after external updating, or even SM based self-updating languages [1]. Two, rSM is *relevant*, meaning that there may exist purely top-down, program call-graph based, query driven methods to determine whether a literal belongs to some model or other. These methods can thus simply return a partial model, guaranteed extendable to a complete one, there existing no

need to compute all models or even to complete models in order to answer a query. Relevance is also crucial for modelling abduction, it being query driven. Three, rSM is cumulative (and two kinds of cumulativity will be considered), so that lemmas may be stored and reused. These and other properties shall be examined in the sequel. These results apply to SM whenever odd loops over negation (OLONs) and infinitely long chains over default negation (ICONs), are absent, thereby establishing significant, not previously known, properties of SM.

Odd Loops Over Negation (OLONs)

In SM, programs such as 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 by *Reductio ad Absurdum*.

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

deploy_WMD ← ~ invade_now invade_now ← deploy_WMD

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

It is an apparently counter-intuitive idea to permit such loops to support a literal's truth value, because it means the truth of the literal is being supported on its own negation, and this seems self-inconsistent. SM does not go a long way in treating such OLON. 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 *Reductio ad Absurdum* (RAA). That is, if assuming a literal false (i.e. its default negation 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 – it requires a true literal to be supported by its rules, i.e. by a rule with true body. The solution proffered by rSM is to extend the notion of support to include reasoning by absurdity, i.e. one supported indeed by those rules creating the odd loop. That is why the single rSM of a ← ~a is {a}.

Example 2: During elections, the prime minister of Italisconia promises to lower taxes as soon as possible, justifying it as inevitable. Indeed, if taxes are not lowered the rich do not invest, the economy cools down, and the country is all the poorer. People thus cannot afford to pay taxes, and these must be lowered anyway:

no_investment ← ~ lower_taxes cool_economy ← no_investment
unaffordable_taxes ← cool_economy lower_taxes ← unaffordable_taxes

Under SM this program has no models. Under rSM lowering taxes is warranted by the single model M={lower_taxes}, and the economy does not cool, etc. These two examples are typical of political *reductio ad absurdum* inevitability arguments.

Example 3: A murder suspect not preventively detained is likely to destroy evidence, and in that case the suspect shall be preventively detained:

likely_to_destroy_evidence(murder_suspect) ← ~ preventively_detain(murder_suspect)
preventively_detain(murder_suspect) ← likely_to_destroy_evidence(murder _suspect)

There is no SM, and a single rSM={ preventively_detain(murder_suspect) }. This jurisprudential reasoning is carried out without need for a murder_suspect to exist now. Should we wish, rSM's *cumulativity* (cf. below) allows adding the model literal as a fact.

Example 4: Some friends are planning their joint vacation. They separately express preference rules Q ← ~R, meaning "I want to go to Q if I cannot go to R", resulting in program P={mountain ← ~beach; beach ← ~travel; travel ← ~mountain}. P has no SMs, but affords three rSMs: {mountain, beach}, {beach, travel}, {travel, mountain}, so all of the friends will be happy if they jointly go on any of these three combination trips.

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 *OLON* is a loop such that the number of default negations in the rule dependency graph path connecting the same literal at both ends is odd.

It may be argued that SM employs OLON as integrity constraints (ICs), and so they should not be resolved; and moreover that the problem remains, in program composition or updating, that unforeseen OLON may appear. We will see below how ICs are dealt with under rSM, separated from the OLONs issue, and so having it both ways, i.e. dealing with OLON *and* ICs.

SM envisages default literals as assumptions that should be maximally true (the Closed World Assumption – 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 its support by program rules. rSM takes this reasoning all the way, but relies on RAA to lend support to the model atoms (minimally) introduced to resolve odd loops and infinitely long support chains.

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, since a third logical value is available: '*undefined'*. In SM, though it's 2-valued, no notion of provability is used, and one resorts to interpreting default negations as assumptions. The rSM view is that assumptions be revised (and hence its name too), in a 2-valued way, if they would otherwise lead to self-inconsistency through odd loops or infinitely long chains.

That rSM resolves the inconsistencies of odd loops of SM (and note these are not contradictions, for there is no explicit negation) does not mean rSM should resolve contradictions. That's an orthogonal problem, whose solutions can be added to different semantics, including rSM.

Infinite chains over negation (ICONs)
It is well-known [5] that SM does not assign semantics either to programs with infinite chains over default negation. We illustrate next that rSM does so.

Example 5: Let P be p(X) ← p(s(X)) p(X) ← ~p(s(X))
P has no SM, but there is one rSM, consisting of p(X) for every X. To see this assume, reasoning by absurd, that p(X) was false for some X; then the two bodies of each clause above would have to be false, meaning that p(s(X)) would be true by the second one; but then, by the first one, p(X) would be true as well, thereby contradicting the default assumption. Hence, by *Reductio ad Absurdum* reasoning, p(X) must be true, for arbitrary X.

The paper's remaining structure: a section on the definition of Revised Stable Models, justification and examples; forthwith, a section on properties of rSM and their contrast with SM's; the last section addresses conclusions, future work, and potential use.

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, and Bi, and default literals not Cj. Often we use '~' for 'not'.

Models are 2-valued and represented as sets of those 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. We will often write S – T to represent the set difference between sets S and T, i.e., S \ T.

Definition 1 (Gelfond-Lifschitz Γ_P operator [2]): 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 $\Gamma_P(I) = J$. Stable Models are the fixpoints of Γ_P, and they do not always exist.

As a shorthand notation, let WFM(P) denote the positive atoms of the Well-Founded Model of P, that is WFM(P) is the least fixpoint of operator Γ_P^2 [6], ie. Γ_P applied twice.

Definition 2 (Sustainable Set): Intuitively, we say a set S is sustainable in P iff any atom 'a' in S does not go against the well-founded consequences of the remaining atoms in S, whenever, S\{a} itself is a sustainable set. The empty set by definition is sustainable . Not going against means that atom {a} cannot be false in the WFM of P∪S\{a}, i.e., 'a' is either true or undefined. That is, it belongs to set $\Gamma_{P\cup S\setminus\{a\}}(WFM(P \cup S\setminus\{a\}))$. Formally, we say S is sustainable iff

$$\forall_{a\in S}\ S\setminus\{a\} \text{ is sustainable} \Rightarrow a\in\Gamma_{P\cup S\setminus\{a\}}(WFM(P \cup S\setminus\{a\}))$$

If S is empty the condition is trivially true.

Definition 3 (Revised Stable Models and Semantics): Let $RAA_P(M) \equiv M - \Gamma_P(M)$. M is a Revised Stable Model of a NLP P, iff:

- M is a minimal classical model, with '~' interpreted as classical negation
- $\exists\ \alpha \geq 2$ such that $\Gamma_P^{\alpha}(M) \supseteq RAA_P(M)$
- $RAA_P(M)$ is sustainable

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

First Condition: *M is a minimal classical model* – A classical model of a NLP is one that satisfies all its rules, where '~' is seen as classical negation and '←' as material implication. Satisfaction means that for every rule body true in the model its head must be true in the model too. Minimality of classical models is required to ensure maximal supportedness (i.e., any true head is supported on a necessarily true body), compatible with model existence.

SMs are supported minimal classical models, and we keep them as a specific case of rSMs. In fact SMs are the special case when there are no inconsistent OLON or ICON. However, not all rSMs are SMs since inconsistent OLON or ICON of an atom 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, and justified through the logical "support" on a specific application of *Reductio Ad Absurdum* (RAA) to that effect.

Example 6: 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 RAA reasoning comes from the requirement to resolve OLON – an issue not dealt with in the traditional SM semantics. In P, $\Gamma_P(\{a\}) = \{\}$ and so $RAA_P(\{a\}) = \{a\} - \{\} = \{a\}$. 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 2-valued, then '~a' 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 classical model with respect to model {a}.

The second condition, explained below, aims at testing the inexorability of a default literal implying its positive counterpart, given the context of the remaining default literals assumed in the candidate model. The $\Gamma_P(M) \subseteq M$ proviso, verified by all minimal models, allows atoms to be minimally added to M over and above those of SMs, since these are defined as $\Gamma_P(SM) = SM$. The additional candidate model atoms are specified by the next condition, i.e. those in $RAA_P(M) = M - \Gamma_P(M)$.

Second Condition: $\exists\, \alpha \geq 2\ \Gamma_P^{\alpha}(M) \supset RAA_P(M)$ – For the sake of explanation, let us first start with a more verbose, but also more intuitive version of this condition:

$$\exists\, \alpha \geq 0\ \Gamma_P^{\alpha}(\Gamma_P(M\text{-}RAA_P(M))) \supseteq RAA_P(M) \quad \text{where } \Gamma_P^{0}(X) = X \text{ for any } X$$

Since $RAA_P(M)=M-\Gamma_P(M)$, the $RAA_P(M)$ set can be understood as the subset of literals of M whose defaults are self-inconsistent, given the rule-supported literals in $\Gamma_P(M)$, the SM part of M. The $RAA_P(M)$ atoms are not obtainable by $\Gamma_P(M)$. The condition states that successively applying the Γ_P operator to $M\text{-}RAA_P(M)$, i.e. to $\Gamma_P(M)$, which is the "non-inconsistent" part of the model or Γ_P rule-supported context of M, we will get a set of literals which, after α iterations of Γ_P, if needed, will get us the $RAA_P(M)$. $RAA_P(M)$ is

thus verified as the set of self-inconsistent literals, whose defaults actually RAA-support their positive counterparts, given the $\Gamma_P(M)$ context.

This is intuitively correct: by assuming the self-inconsistent literals as false, they later appear as true Γ_P consequences. We can simplify this expression to $\exists\ \alpha\geq 0\ \Gamma_P^{\alpha}(\Gamma_P(\Gamma_P(M))) \supseteq RAA_P(M)$. And then to $\exists\ \alpha\geq 2\ \Gamma_P^{\alpha}(M) \supseteq RAA_P(M)$, to obtain the original one. Of course, all SMs comply with this condition because in their case $RAA_P(SM)=\{\}$. So, for SMs all three rSM conditions particularize to the usual definition of $\Gamma_P(SM)=SM$.
The approach to this condition has been inspired by the use of Γ_P and Γ_P^2, in one definition of the Well-Founded Semantics (WFS) [4], to determine the undefined literals. We want to test that the atoms in $RAA_P(M)$ introduced to resolve odd loops and infinite chains, actually lead to themselves through repeated (at least 2) applications of Γ_P, noting that Γ_P^2 is the consequences operator appropriate for odd loop detection (as seen in the WFS), whereas Γ_P is appropriate for even loop SM stability. Since odd loops can have an arbitrary length, repeated (ordinal) applications may be required. Because even loops are stable in just one application of Γ_P, they do not need iteration, which is the case with SMs.

The non-monotonic character of Γ_P, when coupled with the existence of odd loops, may produce temporary spurious elements not in M in the second application of Γ_P in Γ_P^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 $\Gamma_P(M)$ actually are supported by RAA on themselves, given the initial $\Gamma_P(M)$ context. On the other hand, such spurious atoms do not persist: they disappear in the next Γ_P application.

Example 7: a ← ~a, ~b d ← ~a b ← d, ~b

$M_1=\{a\}$, $\Gamma_P(M_1)=\{\}$, $RAA_P(M_1)=\{a\}$, and $M_2=\{b, d\}$, $\Gamma_P(M_2)=\{d\}$, $RAA_P(M_2)=\{b\}$ are the rSMs. Let us see why. $\Gamma_P^2(M_1) = \Gamma_P(\Gamma_P(M_1)) = \Gamma_P(\{\}) = \{a,b,d\} \supseteq \{a\} = RAA_P(M_1)$. Also, $\Gamma_P^2(M_2) = \Gamma_P(\Gamma_P(M_2)) = \Gamma_P(\{d\}) = \{a,b,d\} \supseteq \{b\} = RAA_P(M_2)$; so, both M_1 and M_2 respect the second condition. Since both Models have RAA sets with just one atom and both are undefined in the WFM of the program, the third condition is trivially satisfied for both M_1 and M_2.

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

$M_1=\{a,k\}$, $\Gamma_P(M_1)=\{a,k\}$, $RAA_P(M_1)=\{\}$, $\Gamma_P(M_1)\supseteq RAA_P(M_1)$. M_1 is a rSM and a SM.
$M_2=\{b,k\}$, $\Gamma_P(M_2)=\{b,k\}$, $RAA_P(M_2)=\{\}$, $\Gamma_P(M_2)\supseteq RAA_P(M_2)$. M_2 is a rSM and a SM.
$M_3=\{a,t,i\}$, $\Gamma_P(M_3)=\{a,i\}$, $RAA_P(M_3)=\{t\}$, $\nexists\ \alpha\geq 2\ \Gamma_P^{\alpha}(M_3) \supseteq RAA_P(M_3)$. M_3 is no rSM.
$M_4=\{b,t,i\}$, $\Gamma_P(M_4)=\{b,i\}$, $RAA_P(M_4)=\{t\}$, $\nexists\ \alpha\geq 2\ \Gamma_P^{\alpha}(M_4) \supseteq RAA_P(M_4)$. M_4 is no rSM.

Although M_3 and M_4 are minimal models, the 't' atom in their respective RAA sets is not obtainable by iterations of Γ_P. Simply because '~t', implicit in both $\Gamma_P(M_3)$ and $\Gamma_P(M_4)$, is not conducive to 't' through Γ_P. This is the purpose of the second condition. The attempt to introduce 't' into $RAA_P(M)$ fails because RAA cannot be employed to justify 't'. Indeed, the second condition of the rSM definition is intended to detect negative self-dependencies of atoms in the RAA set. This is clearly not the case of atom 't': it does not depend on itself, let alone on its own negation. For this reason 't' is not a "legitimate" atom to appear in any RAA set.

Example 9: $a \leftarrow \sim b$ $b \leftarrow \sim a$ $c \leftarrow a, \sim c$
$x \leftarrow \sim y$ $y \leftarrow \sim x$ $z \leftarrow x, \sim z$

$M_1 = \{b, y\}$, $M_2 = \{a, c, y\}$, $M_3 = \{b, x, z\}$, $M_4 = \{a, c, x, z\}$, are its rSMs.
$\Gamma_P(M_1) = \{b, y\}$, $\Gamma_P(M_2) = \{a, y\}$, $\Gamma_P(M_3) = \{b, x\}$, $\Gamma_P(M_4) = \{a, x\}$.
$RAA_P(M_1) = \{\}$, $RAA_P(M_2) = \{c\}$, $RAA_P(M_3) = \{z\}$, $RAA_P(M_4) = \{c, z\}$.

In this program we have two even loops (one over 'a' and 'b', and the other over 'x' and 'y'), creating the four possible combinations {a,x}, {a,y}, {b,x}, and {b,y}. Moreover, whenever 'a' is present in a model, the odd loop over 'c' becomes 'active' and so, by RAA we need 'c' to be also present in that model. The same happens for 'x' and 'z'. So the four Minimal Models are {a,c,x,z}, {a,c,y}, {b,x,z}, {b,y}. Since 'c' and 'z' are involved in odd loops, their negations lead to their positive conclusions. It is easy to see that these are the rSMs of P, satisfying the third condition too – there is no dependency on 'z' nor on 'c' and so each of them remains undefined in the Well-Founded Model of P even when the other atom ('c' or 'z') is added as a fact. Note that $RAA_P(M_4)$ is a proper superset of $RAA_P(M_2)$ and of $RAA_P(M_3)$; hence, minimality of RAA sets is *not* a requirement for a rSM.

Example 10: $a \leftarrow \sim b$ $b \leftarrow \sim c$ $c \leftarrow \sim a$

$M_1=\{a,b\}$, $\Gamma_P(M_1)=\{b\}$, $RAA_P(M_1)=\{a\}$, $\Gamma_P^2(M_1) = \{b,c\}$, $\Gamma_P^3(M_1) = \{c\}$, $\Gamma_P^4(M_1)=\{a,c\} \supseteq RAA_P(M_1)$. Since M_1 has an RAA set with just one atom and all of 'a', 'b', and 'c' are undefined in the WFM, the third condition is trivially satisfied. The remaining rSMs, {a,c} and {b,c}, are similar, by symmetry.

Note: In Example 10, it took us 4 iterations of Γ_P to get a superset of $RAA_P(M)$ in a program with an OLON of length 3. In general, a NLP with an OLON of length α will require $\alpha+1$ iterations of the Γ_P operator. Let us see why. First we need to obtain the supported subset of M, which is $\Gamma_P(M)$. The $RAA_P(M)$ set is precisely the subset of M that does not intersect $\Gamma_P(M)$, so under $\Gamma_P(M)$ all literals in $RAA_P(M)$ have truth-value 'false'. Now we start iterating the Γ_P operator over $\Gamma_P(M)$. Since the odd loop has length α, we need α iterations of Γ_P to finally make arise the set $RAA_P(M)$. Hence we need the first iteration of Γ_P to get $\Gamma_P(M)$ and then α iterations over $\Gamma_P(M)$ to get $RAA_P(M)$ leading us to $\alpha+1$. In general, if the odd loop lengths can be decomposed into the primes $\{N_1,\ldots,Nm\}$, then the required number of iterations, besides the initial one, is the product of all the Ni.

The other possible way for a NLP to have no SMs is by having an infinitely long support chain over negation (ICON) even without having any OLONs. An example of such a program first appeared in [5]. It illustrates well the general case for such chains.

Example 11 (François Fage's [5]): $p(X) \leftarrow p(s(X))$ $p(X) \leftarrow \sim p(s(X))$

The grounded version of this program is:

$p(0) \leftarrow p(s(0))$
$p(0) \leftarrow \sim p(s(0))$
$p(s(0)) \leftarrow p(s(s(0)))$
$p(s(0)) \leftarrow \sim p(s(s(0)))$
...

Although P has no Odd-Loop Over Negation, its unique Minimal Classical Model is M = {p(0), p(s(0)), p(s(s(0))), ...}, and complies with the second condition of the definition of rSM. In fact, $\Gamma_P(M) = \{\}$, and $\Gamma_P^2(M) = \Gamma_P(\Gamma_P(M)) = \Gamma_P(\{\}) = M \supseteq RAA_P(M)$. So, M = $RAA_P(M)$. Also, M also complies with the third rSM condition for as soon as one atom $p(s^i(0))$ of $RAA_P(M)$=M is added as a fact, all the other p(0), p(s(0)), …, $p(s^{i-1}(0))$ become true in the Well-Founded Model of the resulting program due to the 'p(X) ← p(s(X))' rules. Moreover, all the other $p(s^j(0))$ – where j>i – atoms remain undefined in the WFM of the resulting program. Hence all atoms in $RAA_P(M)$=M will be elements of $\Gamma_{P\cup R}$ (WFM(P∪R)), for every R⊆$RAA_P(M)$, and $RAA_P(M)$ turns out to be sustainable.

Third Condition: *$RAA_P(M)$ is sustainable*

Let us explain this condition in detail. In a Stable Model there are no elements in the RAA set. This is because there are no actual active OLON or ICON in the program. The only elements we want to admit in any RAA set are those strictly necessary to resolve some actual active OLON or ICON.

The first two conditions of the rSM definition cope with the guarantee that every atom in a rSM is supported, according to a generalized conception of support. There is, however, one additional necessary third condition: the elements in $RAA_P(M)$ must be compatible with each other for $RAA_P(M)$ to be sustainable, in the sense that each of its elements respects the well-founded model obtained by adding to P the remaining elements as facts. That is, every element 'a' of the $RAA_P(M)$ is either true or undefined in the context of all the other atoms of $RAA_P(M)$, but only if the $RAA_P(M)\backslash\{a\}$ set, in turn, verifies the same sustainability condition.

Intuitively, an RAA_P(rSM) set can be incrementally constructed by means of a sequence of sets as follows:

- The first element E_1 in the sequence of sets contains the intersection (which may be empty) of all RAA_P of Minimal Models which respect the second condition of the definition. These are the inevitable and deterministically necessary atoms in any RAA_P(rSM).
- The second element E_2 of the sequence is a singleton set containing one RAA_P atom not in E_1, non-deterministically chosen from some RAA_P of the step before, and which respects all the atoms in the previous set E_1 – i.e., the atom in this singleton set is either true or undefined in the Well-Founded Model of the program in the context of the atoms of the first set (when we add the atoms of the first set to the program as facts).
- The third element E_3 of the sequence contains the intersection of all RAA_P of Minimal Models of P ∪ S which respect the second condition of the RSM definition – where P stands for the original program, and S = $E_1 \cup E_2$. E_3 contains the inevitable and deterministically necessary atoms in any RAA_P(rSM), given the choice in E_2.
- The fourth element E_4 of the sequence is again a singleton set with another non-deterministically chosen atom which, as for the second set, respects the Well-Founded Model of the program in the context of S = $E_1 \cup E_2 \cup E_3$
- Etc.

The sequence construction continues until a Minimal Model is achieved. The RAA_P(rSM) obtained by such sequences comply with the sustainability condition.

Example 12: a ← ~a b ← ~a c ← ~b d ← ~c e ← ~e

With this example we will show how the sequence process can be used to calculate the RAA sets. This program has two Minimal Models: M_1={a,c,e} and M_2={a,b,d,e}. It is easy to verify that both M_1 and M_2 comply with the second condition of the rSM definition. $\Gamma_P(M_1)$={c}, $RAA_P(M_1)$={a,e}; and $\Gamma_P(M_2)$={d}, $RAA_P(M_2)$={a,b,e}. So now we will start the process of creating the acceptable RAA sets. The first element E_1 of the sequence is the intersection of $RAA_P(M_1)$ and $RAA_P(M_2)$ which is {a,e}. Now we add the atoms in E_1={a,e} to the program as facts and calculate the WFM of the resulting program: WFM(P∪{a,e})={a,c,e}. The resulting program P∪{a,e} has two Minimal Models which coincide with M_1 and M_2, but now, under P∪{a,e}, M_2 no longer satisfies the second condition of the rSM definition. In fact, both 'b' and 'd' now become 'false' in the WFM(P∪{a,e}). So the only Minimal Model respecting the second condition, after adding 'a' and 'e' as facts to P is just M_1. $RAA_P(M_1)$ is the only acceptable RAA set (which is sustainable by construction) and hence the unique rSM is M_1.

Example 13: a ← ~b b ← ~c,e c ← ~a e ← ~e, a

We saw that in a rSM, every atom 'a' in RAA must be either true of undefined in the context of $RAA_P(M)\backslash\{a\}$ if $RAA_P(M)\backslash\{a\}$ in turn complies with the same requirement. This happens for any atom 'a' when it does not have only negative dependencies on other RAA_P(M) atoms. Let us see this example in detail. This program has three Minimal Models M_1 = {b,c}, and M_2 = {a,c,e}, and M_3 = {a,b,e}. 'a', 'b', and 'c' and involved in an OLON; and so is 'e'. But the OLON in 'e' is only active when we also have 'a'. So, if we do not have 'a' in a Model we also do not need 'e'; hence the Minimal Model M_1={b,c}. $\Gamma_P(M_1)$={c}, $RAA_P(M_1)$={b}, and $\Gamma_P^4(M_1)$={a,b,e}⊇{b}=$RAA_P(M_1)$, so M_1 respects the second condition. Since all the atoms of the program are undefined in the Well-Founded Model, {b} is sustainable.

M_2={a,c,e}, $\Gamma_P(M_2)$={a}, $RAA_P(M_2)$={c,e}, $\Gamma_P^4(M_2)$={a,b,c,e}⊇{c,e}=$RAA_P(M_2)$. {c} and {e} are sustainable because both 'c' and 'e' are undefined in the WFM of the program. Since 'e'∈$\Gamma_{P\cup\{c\}}$(WFM(P∪{c})) and 'c'∈$\Gamma_{P\cup\{e\}}$(WFM(P∪{e})) we conclude that {c,e} is sustainable and M_2 is also a rSM.

M_3={a,b,e}, $\Gamma_P(M_3)$={}, $RAA_P(M_3)$={a,b,e}, $\Gamma_P^2(M_3)$={a,b,c,e}⊇{a,b,e}=$RAA_P(M_3)$.
In this example, the atom 'a' depends just on '~b', and 'b' is also an element of $RAA_P(M_3)$. This means that if $RAA_P(M_3)\backslash\{a\}$ is sustainable then $RAA_P(M_3)$ is not sustainable and, therefore, M_3 is not a rSM. However, $RAA_P(M_3)\backslash\{a\}$={b,e} is not a sustainable set, and this does not imply $RAA_P(M_3)$ non-sustainability. On the other hand, we have one atom 'b' which is in the $RAA_P(M_3)$ set because it has only positive dependencies on atoms of the $RAA_P(M_3)$ set.

M_3 is a more complex example concerning the third condition. But a quick way of finding out that it also complies with the sustainability requisite is by checking that since 'a' depends negatively on 'b', as soon as we add 'b' as a fact to the program both 'a' and 'e' become immediately false in the WFM(P∪{b}). Hence neither {b,e} nor {a,b} are

sustainable. Since {a,e} is sustainable and 'b'$\in\Gamma_{P\cup\{a,e\}}$(WFM(P$\cup${a,e})) we conclude {a,b,e}=$RAA_P(M_3)$ is sustainable.

Example 14: c ← a, ~c a ← ~b b ← ~a

M_1={b} is a minimal model. $\Gamma_P(M_1)=M_1$, and $RAA_P(M_1)$={}. M_2={a,c} is a minimal model. $\Gamma_P(M_2)$={a}, and $RAA_P(M_2)$={c}. We have as rSMs {b}, the only SM, but also M_2. In fact, 'c' is involved in an odd loop which becomes active when 'a' is true. $\Gamma_P^2(M_2) = \Gamma_P(\Gamma_P(M_2)) = \Gamma_P(\{a\}) = \{a,c\} \supseteq RAA_P(M_2) = \{c\}$. So M_2 respects the second condition. Since the $RAA_P(M_2)$ set consists of just one atom which is undefined in the WFM of the program, and there are no other atoms depending on 'c', adding it to the program as a fact cannot produce any impact one any other atoms of the $RAA_P(M_2)$ set – there are none – and so $RAA_P(M_2)$ is sustainable.

Example 15: a ← ~b b ← ~a c ← a, ~ c c ← b, ~ c d ← b, ~d

There are two Minimal Models for this NLP: M_1 = {a, c} and M_2 = {b, c, d}. They both are Revised Stable Models. Let us see why.
M_1 = {a, c}, $\Gamma_P(M_1) = \Gamma_P(\{a, c\}) = \{a\}$, $RAA_P(M_1) = M_1 - \Gamma_P(M_1) = \{a,c\} - \{a\} = \{c\}$
$\Gamma_P^2(M_1) = \Gamma_P(\Gamma_P(M_1)) = \Gamma_P(\{a\}) = \{a,c\}$, so $\Gamma_P^2(M_1) \supseteq RAA_P(M_1)$. The second condition is satisfied by M_1. The only atoms involved in odd loops are 'c' and 'd'. There are no atoms depending on 'c', so, adding 'c' as a fact to the program will not produce any impact on any other atom of $RAA_P(M_1)$, also because $RAA_P(M_1)$ has just one atom. Let us look now into M_2:
M_2 = {b, c, d}, $\Gamma_P(M_2) = \Gamma_P(\{b, c, d\}) = \{b\}$
$RAA_P(M_2) = M_2 - \Gamma_P(M_2) = \{b, c, d\} - \{b\} = \{c, d\}$
$\Gamma_P^2(M_2) = \Gamma_P(\Gamma_P(M_2)) = \Gamma_P(\{b\}) = \{b, c, d\}$, so $\Gamma_P^2(M_2) \supseteq RAA(M_2)$. The second condition is satisfied by M_2. Similarly to as explained for M_1, no atoms depend on 'c' nor on 'd'; so we conclude that adding any of 'c' or 'd' to the program as a fact will have no impact on the other atom of the RAA set. But let us see it thoroughly. Since both 'c' and 'd' are undefined in the WFM of the program, both {c} and {d} are sustainable. Consider now R = {c} $\subset RAA_P(M_2)$ = {c,d}. The set of positive atoms of P $\cup$ R is WFM(P$\cup$R) = {c}, and $\Gamma_{P\cup R}$ (WFM(P$\cup$R)) = $\Gamma_{P\cup R}(\{c\})$ = {a,b,c,d}, and 'd'$\in${a,b,c,d}. Which means that adding 'c' as a fact to the program does not render 'd' as 'false' in the Well-Founded Model of the resulting program. Doing the same with R={d} we will get the same result for 'c', i.e., 'c'$\in\Gamma_{P\cup R}$(WFM(P$\cup$R))={a,b,c,d}. Hence {c,d} is sustainable and M_2 is a rSM.

Integrity Constraints (ICs)

It may be argued that SM needs to employ OLON for expressing denial ICs, but the problem remains that in program composition unforeseen odd loops may appear, so that not all OLON refer to ICs. rSM can mimic the SM denial ICs by means of OLON involving a programmer chosen reserved literal such as *'falsum*', or simply by means of adding a rule of the form '*falsum* ← IC'. One can then prefer only those rSMs without the '*falsum*' atom. Thus, rSM semantics separates the two issues, having it both ways (cf. Example 13, where one can substitute 'c' for '*falsum*').

Definition 3 (Integrity Constraints) Incorporating denial ICs in a NLP under the Revised Stable Models semantics consists in adding, similarly to SM, a rule in the OLON form

falsum ← an_IC, ~falsum or, more simply, falsum ← an_IC

for each IC, where 'falsum' is a reserved atom chosen by the programmer. The 'an_IC' in the rule stands for a conjunction of literals, which must not be true, and form the IC.

In the case of the OLON introduced this way it results that, whenever 'an_IC' is true, 'falsum' must be in the model. Consequently one can retain only models where 'an_IC' is false, that is those without 'falsum'. Whereas in SM odd loops are used to express ICs, in rSM they can too, but by using the reserved 'falsum' predicate. Or, more simply, by just having the 'an_IC' implying 'falsum'. The OLON form is used by the SMs community to prevent 'an_IC' becoming true, due to the Γ-'instability' of the OLON.

Example 16: In a Middle Region two factions are at odds, and use two sets of reasoning rules. One believes that if terrorism does not stop then oppression will do it and hence become unnecessary. The other believes that if oppression does not stop then terror will do it and hence become unnecessary. Combining them requires two integrity constraints.

oppression_on ← ~ terror_off terror_on ← ~ oppression_off
terror_off ← oppression_on oppression_off ← terror_on

falsum ← oppression_on, oppression_off, ~falsum
falsum ← terror_on, terror_off, ~falsum

So far so good, there is a single joint rSM={oppression_off, terror_off}, and no SM. But introducing either or both of the next two rules, makes it impossible to satisfy the ICs:

oppression_on ← ~ terror_on terror_on ← ~ oppression_on

In this case all the rSMs will contain the atom 'falsum'. Note the difference between rSM and SM semantics concerning this case: under rSM there are still models, they just do not comply with our requirements of not having 'falsum'; whereas under SM semantics there are no models.

Properties of the Revised Stable Models semantics

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

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

SM does not deal with Odd Loops Over Negation nor ICONs, except to prohibit them, and that unfortunately ensures it does not enjoy desirable properties such as *Relevance*. For example, take a program such as:

c ← a, ~c a ← ~b b ← ~a

Although it has a SM={b} it is non-relevant, e.g. in order to find out the truth-value of literal 'a' we cannot just 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 [3] 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 OLONs or ICONs, as these are the ones that may render unacceptable a partial model compatible with the call-graph below a literal. In contradistinction, the even loops can accommodate a partial solution by veering in one direction or the other.

rSM enjoys relevance, by resolving odd loops – and ICONS – in favour of their heads, thus effectively preventing their constraining hold on literals permitting the loop, and so it is potentially amenable to top-down call-graph based query methods. These methods are designed to try and identify whether a query literal belongs to some rSM, and to partially produce an rSM supporting a positive answer. The partial solution is guaranteed extendable to a full rSM because of relevance.

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

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

Example 17: a ← ~b b ← ~a, c c ← a

The single stable model is SM_1={a, c}, $\Gamma_P(SM_1)$=SM_1, and $RAA_P(SM_1)$={}. If 'c ←' is added, then there is an additional SM_2={b, c}, and cumulativity for SM fails because 'a' no longer belongs to the intersection of SMs. There exists also rSM_2={b}, with $\Gamma_P(rSM_2)$={} and $RAA_P(rSM_2)$={b}; so 'c ←' cannot be added as it does not belong to the intersection of all rSMs. This ensures cumulativity for rSMs.

To see why {b} is a rSM note that 'c' in the rule for 'b' partially evaluates into '~b' through the rule for 'a', and that since 'a' is false the rule for 'b' provides an odd loop on 'b'. rSM_2 respects the second condition. In fact, $\Gamma_P^2(rSM_2) = \Gamma_P(\Gamma_P(rSM_2)) = \Gamma_P(\{\}) = \{a,b,c\} \supseteq RAA_P(rSM_2) = \{b\}$. Again, in $RAA_P(rSM_2)$ there is no other atom besides 'b', so there are no dependencies of other atoms of $RAA_P(rSM_2)$ on 'b'. rSM_2 thus respects the third condition since 'b' is undefined in the WFM(P).

Cumulativity [3] pertains to the intersection of models, which formally defines the SM and the rSM semantics. But seldom is this intersection used in practice, and SM implementations are focused instead on computing the set of models.

Another, but similar, second 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 type of cumulativity is common in top-down call-graph oriented query derivation methods, which can exist for rSM because it enjoys 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 those default literals which are true in it. Consequently, if a positive literal A is found true in (finite) 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.

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, complexity analysis, 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.

Having showed that the rSM semantics is equivalent to the SM semantics for programs without OLONs and without ICONS, it is proven that the SM semantics, for these specific "well-behaved" kinds of programs exhibits the Relevancy property as well as guarantee of Model Existence, and Cumulativity. This is, to the best of our knowledge, a new important result about the Stable Models semantics which has never before appeared in the literature.

Another avenue is using rSM, and its extension to default negation in heads, in contrast to SM based ones, as an alternative base semantics for updatable and self-evolving programs [1] so that model *inexistence* after an update may be prevented in a variety of cases. It may be of significance to Semantic Web reasoning, a context where 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 complete groundness and the difficulties it begets for problem representation.

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

Acknowledgements: P. Dell'Acqua, J. Alferes, F. Banti, C. Caleiro, M. Gelfond, M. Knorr, N. Leone, L. Soares, anonymous referees.

References

1. J. J. Alferes, A. Brogi, J. A. Leite, L. M. Pereira. Evolving Logic Programs. In S. Flesca et al. (eds.), 8th European Conf. on Logics in AI (JELIA'02), pp. 50-61, Springer, LNCS 2424, 2002.
2. M. Gelfond, V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski, K. A. Bowen (eds.), 5th Intl. Logic Programming Conf., pp. 1070-1080. MIT Press, 1988.
3. 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.
4. A. van Gelder, K. A. Ross, J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. In J. ACM, 38(3):620-650, 1991.
5. F. Fages. Consistency of Clark's Completion and Existence of Stable Models. In Methods of Logic in Computer Science, vol. 1, pp. 51-60, 1994.
6. A. van Gelder 1993. The Alternating Fixpoint of Logic Programs with Negation. Journal of computer and system sciences 47:185--221.