Supplementary materials for the article

Query Answering in Resource-Based Answer Set Semantics

published in Theory and Practice of Logic Programming

Stefania Costantini
DISIM, Università di L'Aquila
(*e-mail:* `stefania.costantini@univaq.it`)

Andrea Formisano
DMI, Università di Perugia — GNCS-INdAM
(*e-mail:* `formis@dmi.unipg.it`)

submitted May 2016; revised July 2016; accepted August 2016

This appendix contains background material concerning ASP (App. A), Resource-based ASP (App. B), and XSB-resolution (App. C). (All notions have been borrowed from the cited literature). Appendix D contains the proofs of the results in Section 3 of the paper.

A Background on ASP

We refer to the standard definitions concerning propositional general logic programs, as reported, for instance, in (Apt and Bol 1994; Lloyd 1993; Gelfond and Lifschitz 1988). We will sometimes re-elaborate definitions and terminology (without substantial change), in a way which is functional to the discussion.

In the answer set semantics (originally named "stable model semantics"), an answer set program Π (or simply "program") is a finite collection of *rules* of the form $H \leftarrow L_1,\ldots,L_n.$ where H is an atom, $n \geqslant 0$ and each literal L_i is either an atom A_i or its *default negation* $not A_i$. The left-hand side and the right-hand side of rules are called *head* and *body*, respectively. A rule can be rephrased as $H \leftarrow A_1,\ldots,A_m, not A_{m+1},\ldots, not A_n.$ where $A_1,\ldots,A_m$ can be called *positive body* and $not A_{m+1},\ldots, not A_n$ can be called *negative body*. (Observe that an answer set program can be seen as a Datalog program with negation —cf., (Lloyd 1993; Apt and Bol 1994) for definitions about logic programming and Datalog.) A rule with empty body ($n = 0$) is called a *unit rule*, or *fact*. A rule with empty head, of the form $\leftarrow L_1,\ldots,L_n.$, is a *constraint*, and it states that the literals $L_1,\ldots,L_n$ cannot be simultaneously true. A positive program is a logic program including no negative literals and no constraints.

For every atom A occurring in a rule of program Π either as positive literal A or in a negative literal $not A$, we say that A occurs in Π. Therefore, as Π is by definition finite it is possible to determine the set S_Π composed of all the atoms occurring in Π.

In the rest of the paper, whenever it is clear from the context, by "a (logic) program Π" we mean an answer set program (ASP program) Π. As it is customary in the ASP literature, we will implicitly refer to the *ground* version of Π, which is obtained by replacing in all possible ways the variables occurring in Π with the constants occurring in Π itself, and is thus composed of ground atoms, i.e., atoms which contain no variables. We do not consider "classical negation" (cf., (Gelfond and Lifschitz 1991)), nor we consider double negation $not\,not A$. We do not refer (at the moment) to the various useful programming constructs defined and added over time to the basic ASP paradigm.

A program may have several answer sets, or may have no answer set (while in many semantics for logic programming a program admits exactly one "model", however defined). Whenever a program has no answer sets, we will say that the program is *inconsistent*. Correspondingly, checking for consistency means checking for the existence of answer sets.

Consistency of answer set programs is related, as it is well-known, to the occurrence of *negative cycles*, (or negative "loops") i.e. cycles through negation, and to their connections to other parts of the program (cf., e.g., (Costantini 2006)).

To clarify this matter, some preliminary definitions are in order.

Definition A.1 (*Dependency Graph*)
For a ground logic program Π, the dependency graph G_Π is a finite directed graph whose vertices are the atoms occurring in Π (both in positive and negative literals). There is a positive (resp. negative) edge from vertex R to vertex R' iff there is a rule ρ in Π with R as its head where R' occurs positively (resp. negatively) in its body, i.e. there is a positive edge if R' occurs as a positive literal in the body of ρ, and a negative edge if R' occurs in a negative literal $not R'$ in the body of ρ. We say that:

- R depends on R' if there is a path in G_Π from R to R';
- R depends positively on R' if there is a path in G_Π from R to R' containing only positive edges;
- R depends negatively on R' if there is a path in G_Π from R to R' containing at least one negative edge.
- there is an acyclic dependency of R on R' if there is an acyclic path in G_Π from R to R'; such a dependency is even if the path comprises an even number of edges, is odd otherwise.

In this context we assume that R depends on itself only if there exist a non-empty path in G_Π from R to itself. (Note that empty paths are excluded, otherwise each R would always depend -positively- upon itself by definition).

By saying that atom A depends (positively or negatively) upon atom B, we implicitly refer to the above definition.

Definition A.2 (*Cycles*)
A cycle in program Π corresponds to a circuit occurring in G_Π. We say that:

- a positive cycle is a cycle including only positive edges;
- a negative cycle is a cycle including at least one negative edge;
- given a negative cycle C, we say that C is odd (or that C is an odd cycle) if C includes an odd number of negative edges;
- given a positive cycle C, we say that C is even (or that C is an even cycle) if C includes an even number of negative edges;

When referring to positive/negative even/odd cycles we implicitly refer to the above definition.

Below is the formal specification of the answer set semantics, elaborated from (Gelfond and Lifschitz 1988). Preliminarily, we remind the reader that the least Herbrand model of a positive logic program Π can be computed by means of its immediate consequence operator T_Π, that can be defined as follows (the original definition is due to Van Emden and Kowalski). We then introduce the definition of reduct, the Γ operator and finally the definition of answer set. Given a positive program Π and a set of atoms I, let

$$T_\Pi(I) \;=\; \big\{A : \text{there exists a rule } A \leftarrow A_1,\ldots,A_m \text{ in } \Pi \text{ where } \{A_1,\ldots,A_m\} \subseteq I\big\}$$

The T_Π operator always has a unique least fixpoint, that for finite propositional programs is computable in a finite number of steps.

The following definition of (GL-)reduct is due to Gelfond and Lifschitz.

Definition A.3
Let I be a set of atoms and Π a program. The reduct of Π modulo I is a new program, denoted as Π^I, obtained from Π by: 1. removing from Π all rules which contain a negative literal $notA$ such that $A \in I$; and by 2. removing all negative literals from the remaining rules.

Notice that for each negative literal $notA$ which is removed at step 2, it holds that $A \notin I$: otherwise, the rule where it occurs would have been removed at step 1. We can see that Π^I is a positive logic program. Answer sets are defined as follows, via the GL-operator Γ.

Definition A.4 (*The GL-Operator* Γ)
Let I be a set of atoms and Π a program. We denote with $\Gamma_\Pi(I)$ the least Herbrand model of Π^I.

Definition A.5
Let I be a set of atoms and Π a program. I is an *answer set* of Π if and only if $\Gamma_\Pi(I) = I$.

Answer sets form an anti-chain with respect to set inclusion. The answer set semantics extends the well-founded semantics (wfs), formally introduced in (Van Gelder et al. 1991) and then further discussed and characterized (cf. (Apt and Bol 1994) for a survey), that provides a unique three-valued model. The well-founded model $wfs_\Pi = \langle W^+, W^- \rangle$ of program Π is specified by making explicit the set of true and false atoms, all the other atoms implicitly assuming the truth value "undefined". Intuitively, according to the wfs:

- The set W^+ is the set of atoms which can be derived top-down, say, like in Prolog, without incurring in cycles.
- The set W^- is the set of atoms which cannot be derived either because they are not the head of any rule, or because every possible derivation incurs in a positive cycle, or because every possible derivation incurs in some atom which in turn cannot be derived.
- The undefined atoms are those atoms which cannot be derived because every possible derivation incurs in a negative cycle.

Some of the classical models of Π (interpreted in the obvious way as a classical first-order theory, i.e. where the comma stands for conjunction and the symbol $\leftarrow$ stands for implication) can be answer sets, according to some conditions introduced in what follows.

Definition A.6
Given a non-empty set of atoms I and a rule ρ of the form $A \leftarrow A_1,\ldots,A_n, not B_1,\ldots, not B_m$, we say that ρ is *supported* in I iff $\{A_1,\ldots,A_n\} \subseteq I$ and $\{B_1,\ldots,B_m\} \cap I = \emptyset$.

Definition A.7
Given a program Π and a non-empty set of atoms I, we say that I is *supported* w.r.t. Π (or for short Π-supported) iff $\forall A \in I$, A is the head of a rule ρ in Π which is supported in I.

Answer sets of Π, if any exists, are supported minimal classical models of the program. They however enjoy a stricter property, that we introduce below (cf., Proposition A.2).

Definition A.8
Given a program Π and a set of atoms I, an atom $A \in I$ is *consistently supported* w.r.t. Π and I iff there exists a set S of rules of Π such that the following conditions hold (where we say that A is consistently supported via S):

1. every rule in S is supported in I;
2. exactly one rule in S has conclusion A;
3. A does not occur in the positive body of any rule in S;
4. every atom B occurring in the positive body of some rule in S is in turn consistently supported w.r.t. Π and I via a set of rules $S' \subseteq S$.

Note that A cannot occur in the negative body of any rule in S either, since all such rules are supported in I. S is called a *consistent support set* for A (w.r.t. Π and I). Moreover, by condition (ii), different support sets for A may exist, each one including a different rule with head A.

Definition A.9
Given a program Π and a set of atoms I, we say that I is a *consistently supported* set of atoms (w.r.t. Π) iff $\forall A \in I$, A is consistently supported w.r.t. Π and I. We say that I is a *maximal consistently supported* set of atoms (MCS, for short) iff there does not exist $I' \supset I$ such that I' is consistently supported w.r.t. Π. We say, for short, that I is an MCS for Π.

Observe that an MCS can be empty only if it is unique, i.e, only if no non-empty consistently supported set of atoms exists. In both the answer set and the well-founded semantics atoms involved/defined exclusively in positive cycles are assigned truth value *false*. However, the answer set semantics tries to assign a truth value to atoms involved in negative cycles, which are *undefined* under the well-founded semantics (precisely, it succeeds in doing so if the given program Π is consistent). Therefore, for every answer set M, $W^+ \subseteq M$. It is easy to see that:

Proposition A.1
Given the well-founded model $\langle W^+, W^- \rangle$ of program Π, W^+ is a consistently supported set of atoms.

Notice that W^+ is not in general an MCS, as the following proposition holds:

Proposition A.2
Any answer set M of program Π is an MCS for Π.

However, maximal consistently supported sets of atoms are not necessarily answer sets. We introduce some useful properties of answer set semantics from (Dix 1995).

Definition A.10
The sets of atoms a single atom A depends upon, directly or indirectly, positively or negatively, is defined as $\mathit{dependencies\_of}(A) = \{B : A \text{ depends on } B\}$.

Definition A.11
Given a program Π and an atom A, $\mathit{rel\_rul}(\Pi;A)$ is the set of relevant rules of Π with respect to A, i.e. the set of rules that contain an atom $B \in (\{A\} \cup \mathit{dependencies\_of}(A))$ in their heads.

The notions introduced by Def. A.10 and A.11 for an atom A can be plainly generalized to sets of atoms. Notice that, given an atom (or a set of atoms) X, $\mathit{rel\_rul}(\Pi;X)$ is a subprogram of Π. An ASP program can be seen as divided into components, some of them involving cyclic dependencies.

Definition A.12
An answer set program Π is *cyclic* if for every atom A occurring in the head of some rule ρ in Π, it holds that $A \in \mathit{dependencies\_of}(A)$. In particular, Π is *negatively* (resp., *positively*) *cyclic* if some (resp., none) of these dependencies is negative. A program Π in which there is no head A such that $A \in \mathit{dependencies\_of}(A)$ is called *acyclic*.

A cyclic program is not simply a program including some cycle: rather, it is a program where every atom is involved in some cycle. It is easy to see the following.

- An acyclic program has a unique (possibly empty) answer set, coinciding with the set W^+ of true atoms of its well-founded model. Acyclic programs coincide with *stratified* programs in a well-known terminology (Apt and Bol 1994). We prefer to call them 'acyclic' as the notion of strata is irrelevant in the present context.
- A positively cyclic program has a unique empty answer set, coinciding with the set W^+ of true atoms of its well-founded model.
- Negatively cyclic programs have no answer sets and have an empty well-founded model, in the sense that all atoms occurring in such a program are undefined under the well-founded semantics.

In the following, unless explicitly specified by a "cyclic program" (or program component) we intend a negatively cyclic program (or program component, i.e. a subprogram of a larger program). By Definition A.12, there exist programs that are neither cyclic nor acyclic, though involving cyclic and/or acyclic fragments as subprograms, where such fragments can be either independent of or related to each other.

Definition A.13
A subprogram Π_s of a given program Π is *self-contained* (w.r.t. Π) if the set X of atoms occurring (either positively or negatively) in Π_s is such that $\mathit{rel\_rul}(\Pi;X) \subseteq \Pi_s$.

Notice that a subprogram $\Pi_s = \Pi$ is self-contained by definition.

Definition A.14
Given two subprograms Π_{s_1}, Π_{s_2} of a program Π, Π_{s_2} is *on top* of Π_{s_1} if the set X_2 of atoms occurring in the head of some rule in Π_{s_2} is such that $rel\_rul(\Pi; X_2) \subseteq \Pi_{s_2} \cup \Pi_{s_1}$, and the set X_1 of atoms occurring (either positively or negatively) only in the body of rules of Π_{s_2} is such that $rel\_rul(\Pi; X_1) \subseteq \Pi_{s_1}$.[1]

Notice that, by Definition A.14, if Π_{s_2} is *on top* of Π_{s_1}, then X_1 is a *splitting set* for Π in the sense of (Lifschitz and Turner 1994).

Definition A.15
A program obtained as the union of a set of cyclic or acyclic programs, none of which is on top of another one, is called a *jigsaw* program.

Thus any program/component, either acyclic or cyclic or jigsaw, can possibly but not necessarily be self-contained. An entire program is self-contained, but not necessarily jigsaw. We introduce a useful terminology for jigsaw programs which are self-contained.

Definition A.16
Let Π be a program and Π_s a jigsaw subprogram of Π. Then, Π_s is *standalone* (w.r.t. Π) if it is self-contained (w.r.t. Π).

In case we refer to a standalone program Π_s without mentioning the including program Π, we intend Π to be identifiable from the context.

The following property states that a program can be divided into subprograms where a standalone one can be understood as the bottom *layer*, which is at the basis of a "tower" where each level is a jigsaw subprogram standing on top of lower levels.

Proposition A.3
A non-empty answer set program Π can be seen as divided into a sequence of *components*, or layers, $C_1, \ldots, C_n$, $n \geq 1$ where: C_1, which is called the *bottom* of Π, is a standalone program; each component C_i, for $i > 1$, is a jigsaw program which is on top of $C_{i-1} \cup \cdots \cup C_1$.

In fact, the bottom layer (that may coincide with the entire program) necessarily exists as the program is finite, and so does any upper layer. The advantage of such a decomposition is that, by the *Splitting Theorem* introduced in (Lifschitz and Turner 1994), the computation of answer sets of Π can be divided into subsequent phases.

Proposition A.4
Consider a non-empty ASP program Π, divided according to Proposition A.3 into components $C_1, \ldots, C_n$, $n \geq 1$. An answer set S of Π (if any exists) can be computed incrementally as follows:

step 0. Set $i = 1$.
step 1. Compute an answer set S_i of component C_i (for $i = 1$, this accounts to computing an answer set of the standalone bottom component).

[1] This notion was introduced in (Costantini 1995; Lifschitz and Turner 1994).

step 2. Simplify program C_{i+1} by: (i) deleting all rules in which have $not B$ in their body, for some $B \in S_i$; (ii) deleting (from the body of the remaining rules) every literal $not F$ where F does not occur in the head of rules of C_{i+1} and $F \notin S_i$, and every atom E with $E \in S_i$. (Notice that, due to the simplification, C_{i+1} becomes standalone.)

step 3. If $i < n$ set $i = i + 1$ and go to step 1, else set $S = S_1 \cup \cdots \cup S_n$.

All answer sets of Π can be generated via backtracking (from any possible answer set of C_1, combined with any possible answer set of simplified C_2, etc.). If no (other) answer set of Π exists, then at some stage step 1 will fail. An incremental computation of answer sets has also been adopted in (Gebser et al. 2009).

B Background on Resource-Based Answer Set Semantics

The following formulation of resource-based answer set semantics is obtained by introducing some modifications to the original definition of the answer set semantics. Some preliminary elaboration is needed. Following Proposition A.3, a nonempty answer set program Π (that below we call simply "program") can be seen as divided into a sequence of *components*, and, based upon such a decomposition, as stated in Proposition A.4, the answer sets of a program can be computed incrementally in a bottom-up fashion. Resource-based answer sets can be computed in a similar way. Therefore, we start by defining the notion of resource-based answer sets of standalone programs.

The semantic variation that we propose implies slight modifications in the definition of the T_Π and the Γ operator, aimed at forbidding the derivation of atoms that necessarily depend upon their own negation. The modified reduct, in particular, keeps track of negative literals which the "traditional" reduct would remove.

Definition B.1
Let I be a set of atoms and let Π be a program. The *modified reduct* of Π modulo I is a new program, denoted as $\hat{\Pi}^I$, obtained from Π by removing from Π all rules which contain a negative premise $not A$ such that $A \in I$.

For simplicity, let us consider each rule of a program as reordered by grouping its positive and its negative literals, as follows:

$$A \leftarrow A_1, \ldots, A_m,\ not B_1, \ldots, not B_n$$

Moreover, let us define a *guarded atom* to be any expression of the form $A||G$ where A is an atom and $G = \{not C_1, \ldots, not C_\ell\}$ is a possibly empty collection of $\ell \geq 0$ negative literals. We say that A is guarded by the C_is, or that G is a guard for A.

We define a modified T_Π which derives only those facts that do not depend (neither directly nor indirectly) on their own negation. The modified T_Π operates on sets of guarded atoms. For each inferred guarded atom $A||G$, the set G records the negative literals A depends on.

Definition B.2 (*Modified* T_Π)
Given a propositional program Π, let

$$\begin{aligned} T_\Pi(I) \;=\; \Big\{ & A||G_1 \cup \cdots \cup G_r \cup \{not B_1, \ldots, not B_n\} : \text{ there exists a rule} \\ & A \leftarrow A_1, \ldots, A_r, not B_1, \ldots, not B_n \text{ in } \Pi \text{ such that} \\ & \{A_1||G_1, \ldots, A_r||G_r\} \subseteq I \quad \text{and} \quad not A \notin \{not B_1, \ldots, not B_n\} \cup G_1 \cup \cdots \cup G_r \Big\}. \end{aligned}$$

For each $n \geq 0$, let T^n_Π be the set of guarded atoms defined as follows:

$$\begin{array}{rcl} T^0_\Pi & = & \{A||\emptyset : \text{ there exists unit rule } A \leftarrow \text{ in } \Pi\} \\ T^{n+1}_\Pi & = & T_\Pi(T^n_\Pi) \end{array}$$

The *least contradiction-free Herbrand set* of Π is the following set of atoms:

$$\hat{T}_\Pi = \left\{A : A||G \in T^i_\Pi \text{ for some } i \geq 0\right\}.$$

Notice that the least contradiction-free Herbrand set of a (modified reduct of a) program, does not necessarily coincide with the full least Herbrand model of the "traditional" reduct, as its construction excludes from the result those atoms that are guarded by their own negation. We can finally define a modified version of the Γ operator.

Definition B.3 (*Operator* $\hat{\Gamma}$)
Let I be a set of atoms and Π a program. Let $\hat{\Pi}^I$ be the modified reduct of Π modulo I, and J be its least contradiction-free Herbrand set. We define $\hat{\Gamma}_\Pi(I) = J$.

It is easy to see that given a program Π and two sets I_1, I_2 of atoms, if $I_1 \subseteq I_2$ then $\hat{\Gamma}_\Pi(I_1) \supseteq \hat{\Gamma}_\Pi(I_2)$. Indeed, the larger I_2 leads to a potentially smaller modified reduct, since it may causes the removal of more rules.

For technical reasons, we need to consider potentially supported sets of atoms.

Definition B.4
Let Π be a program, and let I be a set of atoms. I is Π-*based* iff for any $A \in I$ there exists rule ρ in Π with head A.

It can be shown (see, (Costantini and Formisano 2015)) that, given a standalone program Π and a non-empty Π-based set I of atoms, and given $M = \hat{\Gamma}_\Pi(I)$, if $M \subseteq I$ then M is a consistently supported set of atoms for Π. Consequently, we have that M is an MCS (cf., Definition A.9) for Π iff there exists I such that $M \subseteq I$, and there is no proper subset I_1 of I such that $\hat{\Gamma}_\Pi(I_1) \subseteq I_1$. We now define resource-based answer sets of a standalone program.

Definition B.5
Let Π be a standalone program, and let I be a Π-based set of atoms. $M = \hat{\Gamma}_\Pi(I)$ is a *resource-based answer set* of Π iff M is an MCS for Π.

It is easy to see that any answer set of a standalone program Π is a resource-based answer set of Π and, if Π is acyclic, the unique answer set of Π is the unique resource-based answer set of Π. These are consequences of the fact that consistent ASP programs are non-contradictory, and the modified T_Π, in absence of contradictions (i.e. in absence of atoms necessarily depending upon their own negations), operates exactly like T_Π. In case of acyclic programs, the unique answer set I is also the unique MCS as the computation of the modified reduct does not cancel any rule, and the modified T_Π can thus draw the maximum set of conclusions, coinciding with I itself.

Being an MCS, a resource-based answer set can be empty only if it is the unique resource-based answer set.

Below we provide the definition of resource-based answer sets of a generic program Π.

Definition B.6
Consider a non-empty ASP program Π, divided according to Proposition A.3 into components $C_1, \dots, C_n$, $n \geq 1$. A resource-based answer set S of Π is defined as $M_1 \cup \dots \cup M_n$ where M_1 is a resource-based answer set of C_1, and each M_i, $1 < i \leq n$, is a resource-based answer set of standalone component C'_i, obtained by simplifying C_i w.r.t. $M_1 \cup \dots \cup M_{i-1}$, where the simplification consists in: (i) deleting all rules in C_i which have $not B$ in their body, $B \in M_1 \cup \dots \cup M_{i-1}$; (ii) deleting (from the body of remaining rules) every literal $not D$ where D does not occur in the head of rules of C_i and $D \notin M_1 \cup \dots \cup M_{i-1}$, and also every atom D with $D \in M_1 \cup \dots \cup M_{i-1}$. (Notice that, due to the simplification, C'_i is standalone.)

Definition B.6 brings evident analogies to the procedure for answer set computation specified in Proposition A.4. This program decomposition is under some aspects reminiscent of the one adopted in (Gebser et al. 2009). However, in general, resource-based answer sets are not models in the classical sense: rather, they are Π-supported sets of atoms which are the wider subsets of some classical model that fulfills non-contradictory support. We can prove, in fact, this result:

Theorem B.1
A set of atoms I is a resource-based answer set of Π iff it is an MCS for Π.

Resource-based answer sets still form (like answer sets) an anti-chain w.r.t. set inclusion, and answer sets (if any) are among the resource-based answer sets. Clearly, resource-based answer sets semantics still extends the well-founded semantics. Differently from answer sets, a (possibly empty) resource-based answer set always exists.

It can be observed that complexity remains the same as for ASP. In fact:

Proposition B.1
Given a program Π, the problem of deciding whether there exists a set of atoms I which is a resource-based answer set of Π is NP-complete.

C XSB-resolution in a Nutshell

Below we briefly illustrate the basic notions of XSB-resolution. An ample literature exists for XSB-resolution, from the seminal work in (Chen and Warren 1993) to the most recent work in (Swift and Warren 2012) where many useful references can also be found. XSB resolution is fully implemented, and information and downloads can be find on the XSB web site, `xsb.sourceforge.net/index.html`.

XSB-resolution adopts *tabling*, that will be useful for our new procedure. Tabled logic programming was first formalized in the early 1980's, and several formalisms and systems have been based both on tabled resolution and on magic sets, which can also be seen as a form of tabled logic programming (c.f. (Swift and Warren 2012) for references). In the Datalog context, tabling simply means that whenever atom S is established to be true or false, it is recorded in a table. Thus, when subsequent calls are made to S, the evaluation ensures that the answer to S refers to the record rather than being re-derived using program rules. Seen abstractly, the table represents the given state of a computation: in this case, subgoals called and their answers so far derived. One powerful feature of tabling is its ability to maintain other global elements of a computation in the "table", such as information about whether one subgoal depends on another, and whether the dependency is through negation. By maintaining this global information, tabling is useful for evaluating logic programs under the well-founded semantics. Tabling allows Datalog programs with negation to terminate with polynomial data complexity under the well-founded semantics.

An abridged specification of the basic concepts underlying XSB-resolution is provided below for the reader's convenience. We refer the reader to the references for a proper understanding. We provide explanations tailored to ground (answer set) programs, where a number of issues are much simpler than the general case (non-ground programs and, particularly, programs with function symbols). For definitions about procedural semantics of logic programs we again refer to (Lloyd 1993; Apt and Bol 1994), and in particular we assume that the reader is to some extent acquainted with the SLD-resolution (Linear resolution with Selection function for Definite programs) and SLDNF-resolution (for logic programs with Negation-as-Failure) proof procedures, which form the computational basis for Prolog systems. Briefly, a ground negative literal succeeds under SLDNF-resolution if its positive counterpart finitely fails, and vice versa it fails if its positive counterpart succeeds. SLDNF-resolution has the advantage of goal-oriented computation and has provided an effective computational basis for logic programming, but it cannot be used as inference procedure for programs including either positive or negative cycles.

XSB-resolution stems from SLS-resolution (Przymusinski 1989; Ross 1992), which is correct and complete w.r.t. the well-founded semantics, via the ability to detect both positive cycles, which make involved atoms false w.r.t. the wfs, and negative cycles, which make the involved atoms undefined. Later, solutions with "memoing" (or "tabling") have been investigated, among which (for positive programs) OLDT-resolution (Tamaki and Sato 1986), which maintains a table of calls and their corresponding answers: thus, later occurrences of the same calls can be resolved using answers instead of program rules. An effective variant of SLS with memoing and simple methods for loop detection is XOLDTNF-resolution (Chen and Warren 1993), which builds upon OLDT. SLG-resolution (Chen and Warren 1996) is a refinement of XOLDTNF-resolution, and is actually the basis of implemented XSB-resolution. In SLG, many software engineering aspects and implementation issues are taken into account. In this context, as we still do not treat practical implementation issues it is sufficient to introduce basic concepts related to SLS and XOLDTNF-resolution.

As done before, let us consider each rule of a program as reordered by grouping its positive and its negative literals, as follows: $A \leftarrow A_1, \ldots, A_m,\ not B_1, \ldots, not B_n$. Moreover, let be given a *goal* of the form $\leftarrow L_1, \ldots, L_k.$, where the L_is are literals, let us consider a *positivistic computation rule*, which is a computation rule that selects all positive literals before any negative ones. These assumptions were originally required by SLS and have been dropped later, but they are useful to simplify the illustration.

The basic building block of SLS-resolution is the *SLP-tree*, which deals with goals of the form $\leftarrow Q$, that form the root of the tree. For each positive subgoal which is encountered, its SLP sub-tree is built basically as done in SLD-resolution. Leaves of the tree can be:

- *dead leaves*, i.e. nodes with no children because either there is no program rule to

apply to the selected atom A, or because A was already selected in an ancestor node (situation which correspond to a positive cycle); in both case the node is *failed*;
- *active leaves*, which are either empty (*successful node*) or contain only negative subgoals.

More precisely, the *Global tree* T for goal $\leftarrow Q$ is built as follows.

- Its root node is the SLP-tree for the original goal.
- Internal tree nodes are SLP-trees for intermediate positive sub-goals.
- *Negation nodes* are created in correspondence of negative subgoals occurring in non-empty active leaves.

The management of negation node works as follows: the negation node corresponding to subgoal *notA* is developed into the SLP-tree for A, unless in case such a node already exists in the tree (negative cycles detection). Then: if some child of a negation node J is a successful tree node, then J is *failed*; if every child of a negation node J is either a failed node or a dead leaf, then J is *successful*.

Any node that can be proved successful or failed is *well-determined*, and any node which is not well-determined is *undetermined*. A *successful branch* of T is a branch that ends at a successful leaf and corresponds to success of the original goal. A goal which leads via any branch to an undetermined node is undetermined. Otherwise, the goal is failed.

It has been proved that a successful goal is composed of literals which are true w.r.t. the wfs, a failed goal includes some literal which is false w.r.t. the wfs, and an undetermined goal includes some literal which is undefined.

XOLDTNF-resolution augments SLS-resolution with tabling and with a simple direct way for negative cycles detection. In the following, given a program Π, let $\mathscr{T}(\Pi)$ be the data structure used by the proof procedure for tabling purposes, i.e. the table associated with the program (or simply "program table"). The improvements of XOLDTNF over SLS are mainly the following.

- The Global tree is split into several trees, one for each call, whose root is an atom A. As soon as the call leads to a result, the "answer", i.e. the truth value of A, is recorded in the table. Only *true* or *undefined* answers are explicitly recorded. Whenever A should occur in a non-root node, it can be resolved only by the answer that has been computed and recorded in $\mathscr{T}(\Pi)$ or that can be computed later. This avoids positive loops. An atom whose associated tree has in the end no answer leaf has truth value *false* because either no applicable program rule exists, or a positive cycle has been encountered.
- For detecting negative cycles the method introduced in the paper is adopted (cf., Definition 3.1).

For Datalog programs, XOLDTNF-resolution is, like SLS-resolution, correct and complete w.r.t. the wfs. Consequently, so are SLG- and XSB-resolution.

D Proofs from the paper

This section contains the proofs of the main Theorems and some preliminary results.

Lemma D.1
Let Π be an acyclic program. RAS-XSB-resolution is correct and complete w.r.t. such a program.

Proof
An acyclic program is stratified and thus admits a two-valued well-founded model (i.e. no atom is undefined) where W^+ coincides with the unique (resource-based) answer set. XSB-resolution is correct and complete w.r.t. such a program. Thus, any literal occurring in Π either definitely succeeds by case 1 of RAS-XSB-resolution or definitely fails by case 2.b of RAS-XSB resolution. Since such cases just resort to plain XSB-resolution, this concludes the proof. $\square$

Lemma D.2
Let Π be a cyclic program. RAS-XSB-resolution is correct and complete w.r.t. such a program.

Proof
Let M be a resource-based answer set of Π. We prove that, for every $A \in M$, query $?{-}\,A$ succeeds under RAS-XSB-resolution. M (which is a maximal consistently supported set of atoms (MCS)) can be obtained by applying the modified immediate consequence operator) to some Π-based set of atoms I. From the application of the modified T_{Π^I} we can trace back a set of program rules from which A can be proved via RAS-XSB-resolution (cases 1 and 3 of Definition of *Success and failure in RAS-XSB-resolution*). Notice first that $T^0_{\Pi^I} = \emptyset$ as a cyclic program includes no fact. (Recall that, by definition, a program is cyclic if each of its heads depends directly or indirectly on itself.) However, Π^I necessarily contains some rule with body including negative literals only, thus leading to a nonempty $T^1_{\Pi^I}$ and determining a final non-empty result of repeated application of T_{Π^I}. For some $i \geq 1$ there will be $A||G \in T^i_{\Pi^I}$ (for a guard G). This means that there exists a rule ρ in Π^I which is applicable, i.e. A does not occur in its body, and $notA$ does not occur in the guard. Let $B_1,\dots,B_n, notC_1,\dots,notC_m$, $n,m \geq 0$ be the body of ρ. Since M is an MCS ρ will be supported in M, i.e. it will hold that $B_i \in M$, $i \leq n$ and $C_j \notin M$, $j \leq m$.

Let us consider the $notC_j$s. It cannot be $C_j \in I$, otherwise, by definition of the modified reduct, rule ρ would have been canceled. Moreover, the C_is are not derived by the modified T_{Π^I} so allowing for the derivation of A. Being the program cyclic, one of the following must be the case for this to happen.

- C_j is not derived by the modified T_{Π^I} (which differs from the standard one only concerning guarded atoms) because it depends positively upon itself and so it is false in every resource-based answer set and in the well-founded semantics. In this case $notC_j$ succeeds by case 3.b of RAS-XSB-resolution: in fact C_j fails by case 2.b since XSB-resolution is correct and complete w.r.t. the well-founded semantics.
- C_j is not derived by the modified T_{Π^I} because it depends negatively upon itself and at some point the derivation incurs in a guard including $notC_j$. In this case, $notC_j$ succeeds either by case 3.c or by case 3.d of RAS-XSB-resolution.

For each of the B_is we can iterate the same reasoning as for A. As noted before, being the program cyclic there are no unit rules, but for M to be nonempty there will exist some rule in Π without positive conditions which is supported in M. Therefore, a RAS-XSB-derivation is always finite. This concludes this part of the proof.

Let us now assume that $?\!-A$ succeeds by RAS-XSB-resolution. We prove that there exists resource-based answer set M such that $A \in M$. We have to recall that a resource-based answer set M is obtained as $M = \hat{\Gamma}_{\Pi}(I)$ where $M \subseteq I$ for some set of atoms I, and that M is an *MCS* for Π. Let us refer to the Definition of *Success and failure in RAS-XSB-resolution* in the paper. Since the program is cyclic, then A succeeds via case 1.b, i.e. there exists a rule ρ in Π (where A does not occur in the body), of the form $A \leftarrow B_1,\dots,B_n, notC_1,\dots,notC_m$ (for $n,m \geq 0$), where all the B_is and all the $notC_j$s succeed via RAS-XSB-resolution. We have to prove that there exists a resource-based answer set M, which is an MCS for Π, where this rule is supported, i.e. it holds that $B_i \in M$ for all $i \leq n$ and $C_j \notin M$ for all $j \leq m$. From the definition of resource-based answer set, M must be obtained from a set of atoms I, where we must assume to select an I such that $A \in I$, $\{B_1,\dots,B_n\} \subseteq I$ and $\{C_1,\dots,C_m\} \cap I = \emptyset$. So, the modified reduct will cancel all rules in Π with $notA$ in their body, while keeping ρ. Thus, we have now to prove that ρ allows the modified T_{Π^I} to add A to M. To this extent, we must consider both the negative and the positive conditions of ρ. Considering the negative conditions, for each the $notC_j$s we can observe that, being Π cyclic, one of the following must be the case.

- $notC_j$ succeeds via either case 3.c or 3.d. It can be one of the following.
  - All rules with head C_j have been canceled by the modified reduct, and so the modified T_{Π^I} cannot derive C_j.
  - There are rules with head C_j which have not been canceled by the modified reduct, and might thus allow the modified T_{Π^I} to derive C_j. Since however Π is cyclic, the application of such a rule will be prevented by the occurrence of $notC_j$ in the guard.
- $notC_j$ succeeds via case 3.b: in this case, being the program cyclic, C_j depends in every possible way positively upon itself. Thus, C_j cannot be derived by the modified T_{Π^I} which, apart from guards, works similarly to the standard immediate consequence operator.

For each of the B_is we can iterate the same reasoning as done for A, and this concludes the proof. $\square$

Lemma D.3
Let Π be a standalone program. RAS-XSB-resolution is correct and complete w.r.t. such a program.

Proof
The result follows from Lemma D.1 and Lemma D.2 as a standalone program is in general a jigsaw program including both cyclic and acyclic components. $\square$

Proof of Theorem 3.1

As a premise, we remind the reader that, according to Definition B.6, for every resource-based answer set M of Π we have $M = M_1 \cup \ldots \cup M_n$, where $C_1 \cup \ldots \cup C_n$ are the components of Π and every M_i is a resource-based answer set of the version of C_i obtained via the simplification specified in the same definition. For every $A \in M$, there exists i, $1 \leq i \leq n$, such that $A \in M_i$.

Let M be a resource-based answer set of Π. We prove that, for every $A \in M$, query $?\!-A$ succeeds under RAS-XSB-resolution. The proof will be by induction.

Induction base. Since C_1 is standalone, then by Lemma D.3 RAS-XSB-resolution is correct and complete w.r.t. M_1 and C_1.

Induction step. Assume that RAS-XSB-resolution is correct w.r.t. subprogram $C_1 \cup \ldots \cup C_i$, $i \leq n$, and its resource-based answer set $M_1 \cup \ldots \cup M_i$. We prove that this also holds for subprogram $C_1 \cup \ldots \cup C_{i+1}$ and its resource-based answer set $M_1 \cup \ldots \cup M_{i+1}$. After the simplification specified in Definition B.6, which accounts to annotating in $\mathscr{T}(\Pi)$ the results of the RAS-XSB derivations of the atoms in M_{i+1}, we have that C_{i+1} becomes standalone, with resource-based answer set M_{i+1}. Then, for $A \in M_{i+1}$ we can perform the same reasoning as for $A \in M_1$, and this concludes the proof. $\Box$

Proof of Theorem 3.2

Given any query $?\!-A$, the set of rules used in the derivation of A constitutes a subprogram Π_A of Π. Therefore, by correctness and completeness of RAS-XSB-resolution there exists some resource-based answer set M_A of Π_A such that, after the end of the derivation, we have $A \in \mathscr{T}(\Pi) \iff A \in M_A$ and $notA \in \mathscr{T}(\Pi) \iff A \notin M_A$ By Modularity of resource-based answer set semantics, there exists some resource-based answer set M of Π such that $M_A \subseteq M$ and therefore $A \in M$. So, let us assume that $?\!-A_1$ succeeds (if in fact it fails, then by correctness and completeness of RAS-XSB-resolution there exist no resource-based answer set of Π including A_1, and by definition of RAS-XSB-resolution the table is left unchanged). For subsequent query $?\!-A_2$ one of the following is the case.

- The query succeeds, and the set of rules used in the derivation of A_2 has no intersection with the set of rules used in the derivation of A_1. Therefore, by Modularity of resource-based answer set semantics we have that $M_{A_1} \cap M_{A_2} = \emptyset$ and there exists resource-based answer set M of Π such that $(M_{A_1} \cup M_{A_2}) \subseteq M$.
- The query succeeds, and the set of rules used in the derivation of A_2 has intersection with the set of rules used in the derivation of A_1. So, some literal in the proof will succeed by cases 1.a and 3.a of RAS-XSB-resolution, i.e, by table look-up. Therefore, by Modularity of resource-based answer set semantics we have that $M_{A_1} \cap M_{A_2} \neq \emptyset$ and there exists resource-based answer set M of Π such that $(M_{A_1} \cup M_{A_2}) \subseteq M$.
- The query fails, and the set of rules attempted in the derivation of A_2 has no intersection with the set of rules used in the derivation of A_1. Therefore, we have that simply there not exists resource-based answer set M such that $A_2 \in M$.
- The query fails, and the set of rules used in the derivation of A_2 has intersection with the set of rules used in the derivation of A_1. So, either some positive literal in the proof will fail by case 1.a of RAS-XSB-resolution or some negative literal in the proof will fail as its positive counterpart succeeds by case 1.a of RAS-XSB-resolution

i.e, in both cases, by table look-up. So, success of A_2 is incompatible with the current state of the table, i.e. with success of A_1. Therefore, by Modularity of resource-based answer set semantics and by correctness and completeness of RAS-XSB-resolution we have that there not exists resource-based answer set M such that $A_1 \in M$ and $A_2 \in M$ and $M_{A_1} \subseteq M$.

The same reasoning can be iterated for subsequent queries, and this concludes the proof. $\square$

References

APT, K. R. AND BOL, R. N. 1994. Logic programming and negation: A survey. *J. Log. Prog. 19/20*, 9–71.

CHEN, W. AND WARREN, D. S. 1993. A goal-oriented approach to computing the well-founded semantics. *J. Log. Prog. 17,* 2/3&4, 279–300.

CHEN, W. AND WARREN, D. S. 1996. Tabled evaluation with delaying for general logic programs. *J. ACM 43,* 1, 20–74.

COSTANTINI, S. 1995. Contributions to the stable model semantics of logic programs with negation. *Theoretical Computer Science 149,* 2, 231–255.

COSTANTINI, S. 2006. On the existence of stable models of non-stratified logic programs. *TPLP 6,* 1-2, 169–212.

COSTANTINI, S. AND FORMISANO, A. 2015. Negation as a resource: a novel view on answer set semantics. *Fundam. Inform. 140,* 3-4, 279–305.

DIX, J. 1995. A classification theory of semantics of normal logic programs I-II. *Fundam. Inform. 22,* 3, 227–255 and 257–288.

GEBSER, M., GHARIB, M., MERCER, R. E., AND SCHAUB, T. 2009. Monotonic answer set programming. *J. Log. Comput. 19,* 4, 539–564.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proc. of the 5th Intl. Conf. and Symposium on Logic Programming*, R. Kowalski and K. Bowen, Eds. MIT Press, Seattle, USA, 1070–1080.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

LIFSCHITZ, V. AND TURNER, H. 1994. Splitting a logic program. In *Proc. of ICLP'94, Intl. Conference on Logic Programming*. MIT Press, Santa Marherita Ligure, Italy, 23–37.

LLOYD, J. W. 1993. *Foundations of Logic Programming*, 2nd ed. Springer, New York, USA.

PRZYMUSINSKI, T. C. 1989. Every logic program has a natural stratification and an iterated least fixed point model. In *Proc. of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, A. Silberschatz, Ed. ACM Press, Philadelphia, USA, 11–21.

ROSS, K. A. 1992. A procedural semantics for well-founded negation in logic programs. *J. Log. Prog. 13,* 1, 1–22.

SWIFT, T. AND WARREN, D. S. 2012. XSB: Extending prolog with tabled logic programming. *TPLP 12,* 1-2, 157–187.

TAMAKI, H. AND SATO, T. 1986. OLD resolution with tabulation. In *Proc. ICLP 1986*, E. Y. Shapiro, Ed. LNCS, vol. 225. Springer, London, UK, 84–98.

VAN GELDER, A., ROSS, K. A., AND SCHLIPF, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM 38,* 3, 620–650.