Query Answering in Resource-Based answer set semantics*

Stefania Costantini[1] and Andrea Formisano[2]

[1] DISIM, Università di L'Aquila, Italy stefania.costantini@univaq.it
[2] DMI, Università di Perugia, Italy formis@dmi.unipg.it

Abstract. In recent work, we defined Resource-Based answer set semantics, which is an extension to traditional answer set semantics stemming from the study of its relationship with linear logic. In this setting there are no inconsistent programs, and constraints are defined "per se" in a separate layer. In this paper, we propose a query-answering procedure reminiscent of Prolog for answer set programs under this extended semantics.

1 Introduction

Answer set programming (ASP) is nowadays a well-established and successful programming paradigm based upon answer set semantics [1, 2, 3], with applications in many areas (cf., e.g., [4, 5, 6] and the references therein). Nevertheless, as noted in [7, 8], few attempts to construct a goal-oriented proof procedure exist. Rather, an ASP-solver is used (see [9]) to find the answer sets, if any exists. This is due to the very nature of the answer set semantics, where a program may admit none or several answer sets, and where the semantics enjoys no locality, or, better, no *relevance* in the sense of [10]: i.e., no subset of the given program can in general be identified, from where the decision of atom A (intended as a goal, or query) belonging or not to some answer set can be drawn. The work of [7] suggests an incremental construction of approximations of answer sets, so as to provide local computations and top-down query answering. A sound and complete proof procedure for the approach is provided. The work of [8] can be used with non-ground queries and with non-ground, and possibly infinite, programs. Soundness and completeness results are proved for large classes of logic programs.

However, another problem related to goal-oriented answer-set-based computation is that of repeated queries. Assume that one would be able to pose a query $?- Q_1$ receiving an answer "yes", to signify that Q_1 is entailed by some answer set of given program Π. Presumably, one might intend subsequent queries to be answered in the same *context*, i.e., a subsequent query $?- Q_2$ might reasonably ask whether some of the answer sets entailing Q_1 also entail Q_2. This might go on until the user explicitly "resets" the context. Such an issue, though reasonable in practical applications, has been hardly addressed up to now, due to the semantic difficulties that we have mentioned.

In recent work, stemming from our research on RASP [11, 12, 13], which is a recent extension of ASP, obtained by explicitly introducing the notion of *resource*, we

* This research will be presented at the ASPOCP14 workshop. This research is partially supported by GNCS-13 and GNCS-14 projects.

proposed in [14] a comparison between RASP, ASP and linear logic [15]. For establishing this correspondence, we introduced a RASP and linear-logic modeling of default negation as understood under the answer set semantics. This led to the definition of an extension to the answer set semantics which we called *Resource-Based answer set semantics* (RAS) [16]. This extension finds an alternative equivalent definition in a variation of the auto-epistemic logic characterization of answer set semantics discussed in [17]. In resource-based answer set semantics we have no inconsistent programs, and constraints are defined "per se" in a separate layer.

This allows us to propose here top-down procedure for the new semantics which, via a form of tabling, also provides contextualization. Differently from [7], this procedure does not require actual incremental answer set construction when answering a query. Rather, it exploits the fact that resource-based answer set semantics enjoys the property of relevance [10] (where answer set semantics does not), which guarantees that truth value of an atom can be established on the basis of the subprogram it depends upon, and thus allows for top-down computation starting from a query. As answer set semantics and resource-based answer set semantics extend the well-founded semantics [18], we take as a starting point XSB-resolution [19, 20], an efficient, fully described and fully implemented procedure which is correct and, for the class programs considered in the answer set semantics, always complete w.r.t. the well-founded semantics of given program. In this paper we do not provide the full detail of the proposed procedure, which we call RAS-XSB-resolution. In fact, this would imply suitably extending and reworking all definitions related to XSB. We rather lay the foundation, however with a precision and formality that should be sufficient to allow such a refinement as the next step. We also provide formal properties of the proposed procedure.

Notice that RAS-XSB resolution can be used for "traditional" answer set programming under the software engineering discipline of dividing the program into a consistent "base" level and a "top" level including constraints. Therefore, even to readers not particularly interested in the new semantics, the paper proposes a full top-down query-answering procedure for ASP, though applicable with previously mentioned (reasonable) limitation. With respect to top-down procedure proposed [8], we do not aim at managing function symbols (and thus programs with infinite grounding), so under this extent our work is more limited. However, we get correctness and completeness for every program (under the new semantics).

In the rest of the paper, after a short introduction of the answer set semantics we summarize resource-based answer set semantics. We then proceed to present the original contributions of this paper, that consist in introducing some useful properties of RAS, and in the definition of RAS-XSB-resolution. Background notions which are useful for a better understanding and proof of the main theorem are provided in the extended version of this paper, available online [21].

2 Background on ASP

In the Answer Set Semantics (originally named "stable model semantics"), a (logic) program Π (cf., [1]) is a collection of *rules* of the form $H \leftarrow L_1, \ldots, L_n.$ where H is an atom, $n \geq 0$ and each literal L_i is either an atom A_i or its *default negation not* A_i.

An answer set program can be seen as a Datalog program with negation (cf. [22, 23] for definitions about logic programming and Datalog). In what follows, unless explicitly differently specified we refer to *ground* programs, i.e., programs not containing variables. Below is the specification of the answer set semantics, reported from [1].

Definition 1. *Given ASP program Π and set of atoms I, the Γ operator performs the following steps: (a) Computes the* reduct *Π^I of Π, by:*
(1) removing from Π all rules which contain a negative premise not A such that $A \in I$;
(2) removing from the remaining rules those negative premises not A such that $A \notin I$;
notice that Π^I is a positive logic program.
(b) Computes the Least Herbrand Model of Π^I, denoted as $\Gamma_\Pi(I)$.

Definition 2. *A set of atoms I is an answer set for a program Π iff $\Gamma_\Pi(I) = I$.*

Answer sets are minimal supported models of the program interpreted in the obvious way as a first-order theory ($\leftarrow$ stands for implication, comma for conjunction and *not* for classical negation). It will be useful in what follows to consider a simple property of Γ_Π (see [24]): if M is a minimal model of Π, then, $\Gamma_\Pi(M) \subseteq M$.

In the answer set semantics, a rule of the form $\leftarrow L_1, \ldots, L_n.$ is called *constraint*, and states that the L_is cannot be all true w.r.t. any answer set. It is rephrased into a standard rule $Q \leftarrow$ *not* $Q, L_1, \ldots, L_n.$ with Q fresh atom, as a contradiction on Q leads to *inconsistency*, i.e., non-existence of answer sets (which in fact can in general be several, one, or none) unless one of the L_is is false.

In this paper we refer for lack of space to the basic version of the answer set semantics. Therefore, we do not consider the various extensions and useful programming constructs that have appeared in the wide existing literature about ASP.

3 Resource-Based answer set semantics

In this section we introduce a formal definition of resource-based answer set semantics, which is needed in order to be able to define the proposed proof procedure and prove its properties. However, some preliminary observations are in order so as to explain why resource-based answer set semantics is reasonable, and might possibly be adopted as a *proper extension* of the answer set semantics. As it is well-known, the answer set semantics *extends* the well-founded semantics [18], which assigns to a logic program Π a unique, three-valued model, called well-founded model and denoted by $WFS(\Pi) = \langle W^+, W^- \rangle$, where W^+ and W^- are disjoint. In particular, W^+ is the set of atoms deemed true, W^- is the set of atoms deemed false, while atoms belonging to neither set are deemed *undefined.* Atoms with truth value 'undefined' under the well-founded semantics are exactly the atoms which are of interest for finding the answer sets, and are, in particular, atoms involved in *negative cycles*, i.e. cycles through negation (as extensively discussed, e.g., in [24, 25, 26] and in the references therein).

In particular, the answer set semantics selects some of the (two-valued) classical models of given program so as for each atom A which is true w.r.t. an answer set M, two conditions hold: (i) A is supported in M by some rule of the given program; (ii) the support of A does not depend (directly or indirectly) upon the negation of another true

atom, including itself. For *even cycles*[3] such as $\{e \leftarrow \mathit{not}\ f.\ f \leftarrow \mathit{not}\ e.\}$, two answer sets can be found, namely $\{e\}$ and $\{f\}$, respecting both conditions. This extends to wider program including such cycles. For odd cycles (such as unary odd cycles of the form $\{p \leftarrow \mathit{not}\ p.\}$ and ternary odd cycles of the form $\{a \leftarrow \mathit{not}\ b.\ b \leftarrow \mathit{not}\ c.\ c \leftarrow \mathit{not}\ a.\}$) it is not possible to assign truth values to their composing atoms in classical models. Thus, under the answer set semantics a program including such cycles is *inconsistent*, i.e., it has no answer sets[4]. In a sense, the answer set semantics is still three-valued, as sometimes it is able to assign truth value to atoms, and sometimes (when the program is inconsistent) leaves them all undefined.

Resource-based answer set semantics is able to cope with any kind of cycle, and always assigns a truth value to all atoms while fulfilling conditions (i) and (ii). This by resorting to *supported subsets* of classical models. For the unary odd cycle, p is deemed to be false because were it true, it would depend upon the negation of a true atom (itself). The ternary odd cycle has the three resource-based answer sets $\{a\}$, $\{b\}$ and $\{c\}$. Taking for instance $\{a\}$, atom b must be false to fulfill condition (i), and atom c must be false to fulfill condition (i) for itself and (ii) for a.

Logical foundations of resource-based answer set semantics are discussed in depth in [27]. A characterization can be obtained by elaborating the auto-epistemic-logic characterization of answer set semantics discussed in [17]. Intuitively (for precise definitions please refer to [27]), a rule $A \leftarrow\ A_1, \ldots, A_n, \mathit{not}\ B_1, \ldots, \mathit{not}\ B_m$ can be seen as standing for $A_1 \wedge \ldots \wedge A_n \wedge L\neg B_1 \wedge \ldots \wedge L\neg B_m \wedge L\dot{A} \supset A$. Where $L\,\varphi$ can be understood as "φ is believed", or also "we assume φ", and $\dot{A}$ can be understood as "one intends to prove A". The overall reading is that A is derived whenever the positive conditions hold, we assume that the negative conditions hold as well, and we assume that we indeed intend to prove A. Clearly, we have to state that $A \wedge L\neg A \supset \bot$ (if we have A we cannot believe its negation) and $L\dot{A} \wedge L\neg A \supset \bot$ (we cannot intend to prove A if we believe its negation). This characterization can be transposed into plain ASP by interpreting modal literals as fresh atoms. The answer sets (which are among the classical models) of the transposition Π' of a given program Π coincide, when removing fresh atoms, with the resource-based answer sets of Π. Every program admits at least one (possibly empty) resource-based answer set.

The denomination of resource-based answer set semantics stems from the linear logic formulation of ASP that we proposed in [14, 16], which constituted the original inspiration for the new semantics. This formulation interprets negation *not* A of atom A as a resource that is unlimitedly available unless A is proved. Therefore, *not* A can be freely used whenever needed but: (1) *not* A becomes unavailable if A is proved; (2) whenever *not* A has been used, A can no longer be proved. For unary odd cycles such as $\{p \leftarrow \mathit{not}\ p.\}$ in the linear logic formulation upon the attempt of using *not* p for proving p, by condition (2) p becomes no longer provable (and thus it is false). Similarly for ternary odd cycles. Thus, under the resource-based answer set semantics a 3-atoms odd cycle is interpreted as an exclusive disjunction, exactly like 2-atoms even cycles. In the generate-and-test perspective which is the basis of the ASP programming methodology,

[3] Even (resp. odd) cycles are cycles involving an even (resp. odd) number of negative dependencies, cf., e.g., [24, 25, 26] for precise definitions.

[4] Unless "handles" are provided from other parts of the program, see [24, 25, 26] for details.

even cycles are commonly used to generate the search space. Thus, our new semantics provides some additional flexibility in this sense.

Resource-based answer set semantics is significantly different from other valuable semantic approaches aimed at managing odd cycles, such as [28, 29] and [30, 31]. Such a semantics can be characterized, similarly to traditional answer set semantics, by means of the Γ operator (cf. Definition 1 in Sect. 2). In fact, the resource-based interpretation of negation requires that the negation of an atom (seen as a resource) that has been proved, becomes unavailable: the effect of Γ is in fact exactly that of eliminating rules that make such use of negation.

For providing the formal definition, some preliminary consideration is needed. As discussed in [21], a nonempty answer set program Π (that below we call simply "program") can be seen as divided into a sequence of *components*, or layers, $C_1, \ldots, C_n$, $n \geq 1$, where each C_i is the union of a set of cyclic or acyclic subprograms (subcomponents) independent of each other (with no atoms in common); each subcomponent of C_1, which is called the *bottom* of Π, is standalone, i.e., the atoms occurring therein do not depend upon other parts of the program; each subcomponent of C_i, $i > 1$, is on top of some subcomponent of C_{i-1}, i.e., the atoms occurring therein depend upon atoms occurring in C_{i-1}. For the formal definition of cyclic, acyclic on-top and standalone subprograms, refer to [21]. Based upon such a decomposition, as first discussed in [32], the answer sets of a program can be computed incrementally in a bottom-up fashion.

Proposition 1. *Consider a nonempty ASP program Π, divided into components $C_1, \ldots, C_n$, $n \geq 1$. An answer set S of Π (if any exists) can be computed incrementally by means of the following steps:*

0. *Set $i = 1$.*
1. *Compute an answer set S_i of component C_i (for $i = 1$, this accounts to computing an answer set of the bottom component).*
2. *Simplify program C_{i+1} by: (i) deleting all rules in which have not B in their body, $B \in S_i$; (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}, $F \notin S_i$, and every atom E with $E \in S_1$. Notice that, once simplified, C_{i+1} becomes standalone.*
3. *If $i < n$ set $i = i + 1$ and go to step 1, else set $S = S_1 \cup \ldots \cup S_n$.*

Resource-based answer sets can be computed in a similar way. We start by defining the notion of resource-based answer sets of a given standalone program. In particular, they are obtained from some of its minimal models, specifically from the Π-based minimal models:

Definition 3. *A Π-based minimal model I of an ASP program Π is either the empty set (in case it is the unique minimal model), or a nonempty minimal model such that $\forall A \in I$, there is a rule in Π with head A, where A does not occur positively in the body.*

The restriction to Π-based minimal models is due to the fact that resource-based answer sets are supported sets of atoms. Thus, we aim at avoiding unsupported minimal models, such as, for sample one-rule program $a \leftarrow not\ c$, the minimal model $\{c\}$. Being Π-based is only a prerequisite for supportedness which however will be guaranteed by other conditions. Below we provide a variation of the answer set semantics that defines resource-based answer sets.

Definition 4. *Let Π be a standalone program, and let I be a Π-based minimal model. M is a resource-based answer set of Π iff $M = \Gamma_\Pi(I)$ (we remind the reader that, for any model I and program Π, $\Gamma_\Pi(I) \subseteq I$).*

We are now ready to define resource-based answer sets of a generic program Π.

Definition 5. *Consider a nonempty ASP program Π, divided into components $C_1, \ldots, C_n$, $n \geq 1$. A resource-based answer set S of Π is defined as $M_1 \cup \ldots \cup M_n$ 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. $S = M_1 \cup \ldots \cup M_{i-1}$, where the simplification consists in: (i) deleting all rules in C_i which have not B in their body, $B \in S$; (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 $D \notin S$, and also every atom D with $D \in S$ (notice in fact that, once simplified, C_{i+1} becomes standalone and therefore Definition 4 can be applied).*

The above definition brings clear analogies to the procedure for answer set computation specified in [21]. Therefore, it is easy to see that, for consistent ASP programs, answer sets are among resource-based answer sets. Proposed program decomposition is also reminiscent of the one adopted in [7]. However, in general, resource-based answer sets are not models in the classical sense: rather, they are sets of atoms which are subsets of some minimal model, and are supported (similarly to answer sets, which are minimal supported models): in fact, from the above definitions it can be easily seen that for every atom A in a resource-based answer set M, there exists a rule with head A, and body which is true w.r.t. M. Non-empty resource-based answer sets still form an anti-chain w.r.t. set inclusion.

We now explain by means of an example why the incremental construction of resource-based answer set is needed. Let Π^E be the following:

$a \leftarrow \mathit{not}\ p.$ $\quad$ $p \leftarrow \mathit{not}\ p.$ $\quad$ $q \leftarrow e.$ $\quad$ $e \leftarrow \mathit{not}\ q.$

Suppose to apply Definition 4 directly to the overall program. It admits a unique Π^E-based minimal model $S = \{p, q\}$, and we have $\Gamma_{\Pi^E}(S) = \emptyset$. This is reasonable for p and q: in fact, they depend upon their own negation, so in our perspective there is not "enough" negation to prove them, thus they must be deemed to be false. It is reasonable also for e, which is involved (though through negation) in a positive circularity. It is however not reasonable for a, which depends upon negation of a false atom. However, according to Definition 5, we divide Π^E into a standalone bottom component C_1, consisting of the last three rules, with $M_1 = \emptyset$ as the unique resource-based answer set, and a top component C_2 consisting of the first rule $a \leftarrow \mathit{not}\ p$: after simplification, C_2' is simply fact a, unique resource-based answer set $M_2 = \{a\}$, which coincides with the unique resource-based answer set of the overall program, thus meeting the intuition.

We have called the new semantics Resource-Based Answer Set semantics (RAS), w.r.t. AS (Answer Set) semantics. Differently from answer sets, a (possibly empty) resource-based answer set always exists. Complexity of RAS semantics is however higher than complexity of AS semantics: in fact, [33] proves that deciding whether a set of formulas is a minimal model of a propositional theory is co-NP-complete. Clearly, checking whether a minimal model I is Π-based and computing $\Gamma_{\Pi_s}(I)$ has polyno-

mial complexity. Then, checking whether a set of atom I is a resource-based answer set of program Π is co-NP-complete.

In resource-based answer set semantics there are no inconsistent programs. This means that constraints cannot be modeled (as done in ASP) in terms of odd cycles. Hence, they have to be modeled explicitly. Without loss of generality we will assume in the rest of the paper the following simplification concerning constraints. Each constraint $\leftarrow L_1, \ldots, L_k$, $k > 0$, where each L_i is a literal, can be rephrased as simple constraint $\leftarrow H$, where H is a fresh atom, plus rule $H \leftarrow L_1, \ldots, L_k$ to be added to given program Π. We will from now on implicitly consider the version of Π enriched by such rules.

Definition 6. *Let Π be a program and $\{C_1, \ldots, C_k\}$ be a set of constraints, each C_i in the form $\leftarrow H_i$. A resource-based answer set M for Π is* admissible *if it fulfills all constraints, i.e., if for all $i \leq k$, $H_i \notin M$. M is* admissible *w.r.t. a single constraint C_j if $H_j \notin M$.*

4 Properties of Resource-Based answer set semantics

It is relevant, also for what follows, to evaluate RAS with respect to general properties of semantics of logic programs introduced in [10], that we recall below.

Definition 7. *The sets of atoms a single atom A depends upon, directly or indirectly, positively of negatively, is defined as $dependencies\_of(A) = \{B : A \text{ depends on } B\}$.*

The former definition is provided with some approximation, as dependencies should be formally checked on the *dependency graph* of given program [22, 23].

Definition 8. *Given a program Π and an atom A, $rel\_rul(\Pi; A)$ is the set of relevant rules of Π with respect to A, i.e. the set of rules that contain an atom $B \in dependencies\_of(A)$ in their heads.*

Note that the notions introduced by Definitions 7 and 8 for an atom A are plainly generalized to any set X of atoms. Notice, moreover, that given an atom (or a set of atoms) X, $rel\_rul(\Pi; X)$ is a subprogram of Π.

Definition 9. *Given any semantics SEM and a ground program Π,* Relevance *states that for all literals L it holds that $SEM(\Pi)(L) = SEM(rel\_rul(\Pi; L))(L)$.*

Relevance implies that the truth value of any literal under that semantics in a given programs is determined solely by the subprogram consisting of the relevant rules. The answer set semantics does not enjoy relevance [10]. This is one reason for the lack of goal-oriented proof procedures. Instead, it is easy to see that

Proposition 2. *Resource-based answer set semantics enjoys Relevance.*

Resource-based answer set semantics, like most semantics for logic programs with negation, enjoys *Reduction*, which simply assures that the atoms not occurring in the heads of a program are always assigned truth value 'false'. Resource-based answer set semantics also enjoys *Modularity* [10] (where the reduct Π^M of program Π w.r.t. set of atoms M is recalled in Definition 1.):

Definition 10. *Given any semantics SEM, a ground program Π let $\Pi = \Pi_1 \cup \Pi_2$ where for every atom A occurring in Π_2, $rel\_rul(\Pi; A) \subseteq \Pi_2$. SEM enjoys* Modularity *if $SEM(\Pi) = SEM(\Pi_1^{SEM(\Pi_2)} \cup \Pi_2)$.*

We can in fact prove the following proposition:

Proposition 3. *Given a ground program Π let $\Pi = \Pi_1 \cup \Pi_2$, where for every atom A occurring in Π_2, $rel\_rul(\Pi; A) \subseteq \Pi_2$. A set M of atoms is a resource-based answer set of Π iff there exists a resource-based answer set S of Π_2 such that M is a resource-based answer set of $\Pi_1^S \cup \Pi_2$.*

Modularity is an important property, that also impacts on constraint checking, i.e., on the check of admissibility of resource-based answer sets. Considering, in fact, a set of constraints $\{C_1, \ldots, C_n\}$, $n > 0$, each C_i in the form $\leftarrow H_i$, and letting for each $i \leq n$ $rel\_rul(\Pi; H_i) \subseteq \Pi_2$, from Prop. 3 it follows that, if a resource-based answer set X of Π_2 is admissible (in terms of Definition 6) w.r.t. $\{C_1, \ldots, C_n\}$, then any resource-based answer set M of Π such that $X \subseteq M$ is also admissible w.r.t. this set of constraints. In particular, Π_2 can be identified in relation to a certain query, i.e.:

Definition 11. *Given a program Π, a constraint $\leftarrow H$ associated to Π is* relevant for query $?- A$ *if $rel\_rul(\Pi; A) \subseteq rel\_rul(\Pi; H)$.*

5 A Proof Procedure for RAS

As said before, the answer set semantics *extends* the well-founded semantics. Resource-based answer set semantics still extends the well-founded semantics, as it still deals with assigning a truth value to atoms which are undefined under the this semantics: however, it is able to cope with odd cycles that the answer set semantics interprets as inconsistencies. Assuming to devise a query-answering device for ASP, query $?- A$ to ASP program Π may be reasonably expected to succeed or fail if A belongs to W^+ or W^- respectively, but how to find an answer if A is undefined because it is involved in a negative circularity remains to be understood.

An additional problem with answer set semantics is that query $?- A$ might *locally* succeed, but still, for the lack of relevance, the overall program may not have answer sets (i.e., the program is *inconsistent*). In resource-based answer set semantics instead, there are no inconsistent programs and every program has at least one (possibly empty) resource-based answer set: each of them taken singularly is then admissible or not w.r.t. the integrity constraints. This allows one to defer constraint checking in case the proof of query A succeeds. In this section, we present and discuss the foundations of a proof procedure for logic programs under resource-based answer set semantics.

We take as a starting point a well-established proof procedure for the well-founded semantics, namely XSB-resolution. An ample literature exists for XSB-resolution, from the seminal work in [20] to the most recent work in [19] 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`.

For lack of space, here we do not describe XSB-resolution in detail. We provide in [21] some definitions and results useful for a general understanding. In the rest of this

section, we proceed to illustrate how we mean to extend XSB in order to cope with undefined atoms.

XSB-resolution [19] adopts *tabling*, that will also be useful in what follows. 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. [19] 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. The essential idea is that global information about dependencies is used to determine the truth value of literals that do not have a derivation. If such literals are involved in a cyclic dependency through negation, they are undefined under WFS; if not, the literals belong to an unfounded set and are false in WFS. In fact, it can be shown that tabling allows Datalog programs with negation to terminate with polynomial data complexity under the well-founded semantics.

We will now define the foundations of a top-down proof procedure for resource-based answer set semantics, which we call RAS-XSB-Resolution. The procedure has to cope with the fact that there are atoms which are involved in negative circularities, and must be assigned a truth value according to some resource-based answer set. We build upon XSB-Resolution, which is by no means elementary, so we refer the reader to the references for a proper understanding. An abridged specification is provided below for the reader's convenience, based upon preliminary definition reported in [21]. In order to give an intuitive idea, we resort, in fact, to the following "naive" formulation, relying upon general definitions reported in [22, 23].

Definition 12 (A "naive" XSB-resolution). *Given a program Π, let $\mathcal{T}able(\Pi)$ be the data structure used by the proof procedure for tabling purposes, i.e., the table associated with the program (or simply "program table"). Given a query $?- A$, the list of current subgoals is initially set to $\mathcal{L}_1 = \{A\}$ and $\mathcal{T}able(\Pi)$ is initialized to be the empty set. If in the construction of a proof-tree for $?- A$, a literal L_{i_j} is selected in the list of current subgoals $\mathcal{L}_i$, we have that: if L_{i_j} definitely succeeds (in case of a negative literal $L_{i_j} = not\ B$, it definitely succeeds if B definitely fails) then we take L_{i_j} as proved and proceed to prove $L_{i_{j+1}}$ after the related updates to the program table. Otherwise, we have to backtrack to previous list $\mathcal{L}_{i-1}$ of subgoals. Success and failure determine suitable modifications to $\mathcal{T}able(\Pi)$.*

On Datalog programs XSB is correct and complete, therefore, under XSB-resolution, atom A definitely succeeds iff $A \in W^+$ and definitely fails iff $A \in W^-$. Note that definite failure occurs not only when at some point there is an atom not defined by any rules, but also whenever an atom depends positively in any possible (even

indirect) way upon itself. In our extension, we take the above result as a starting point for success and failure.

In order to represent the notion of negation as a resource, we initialize the program table prior to posing queries and we manage the table during a proof so as to state that: the negation of any atom which is not a fact is available unless this atom has been proved; the negation of an atom which has been proved becomes unavailable; the negation of an atom which cannot be proved is always available.

Definition 13 (Table Initialization in RAS-XSB-Resolution). *Given a program Π and an associated table $\mathcal{T}able(\Pi)$,* Initialization *of $\mathcal{T}able(\Pi)$ is performed by inserting, for each atom A occurring as the conclusion of some rule in Π, a fact $yesA$ (where $yesA$ is a fresh atom).*

The meaning of $yesA$ is that the whole amount of A's negation is still available. If $yesA$ is present then A can possibly succeed. Success of A "absorbs" $yesA$ and prevents *not* A from success. Resource-based answer set semantics in fact dictates that proving A consumes the whole amount of A's negation. $\mathcal{T}able(\Pi)$ will evolve during a proof into subsequent "knowledge states". In the following, without loss of generality we can assume that a query is of the form $?\!- A$, where A is an atom. A proof of query $?\!- A$ is performed by XSB-resolution, though with the following additive modifications.

Definition 14 (Success and failure in RAS-XSB-Resolution). *Given program Π and its associated table $\mathcal{T}able(\Pi)$, notions of success and failure and of modifications to $\mathcal{T}able(\Pi)$ are extended as follows with respect to XSB-Resolution.*

(1) Atom A succeeds if one of the following is the case:
  (a) A is present in $\mathcal{T}able(\Pi)$.
  (b) Fact $yesA$ is present in $\mathcal{T}able(\Pi)$, and there exists in Π either fact A or a rule of the form $A \leftarrow L_1, \ldots, L_n$, $n > 0$, such that every L_i, $i \geq n$, succeeds. Definite success of A is a particular case.

  In consequence of success of A, fact A is added to $\mathcal{T}able(\Pi)$ (if not already present), and fact $yesA$ is removed.

(2) Atom A fails if one of the following is the case:
  (a) Fact $yesA$ is not present in $\mathcal{T}able(\Pi)$, and therefore A is unprovable.
  (b) A definitely fails.
  (c) There exists no rule of the form $A \leftarrow L_1, \ldots, L_n$, $n > 0$, such that every L_i succeeds, as one of the following is the case:
    (i) Some positive literal, among $L_1, \ldots, L_n$, fails.
    (ii) Some negative literal, among $L_1, \ldots, L_n$, fails.
    (iii) Any possible derivation of some of the L_is, $i \leq n$, incurs into not A directly, i.e., not through layers of negation.
    (iv) Any possible derivation of some of the L_is $i \leq n$, incurs into A through layers of negation that do not involve not A.

  In cases (iii) and (iv) we say that A is forced to failure. *In consequence of failure of A, fact $yesA$ is removed from $\mathcal{T}able(\Pi)$ (if present). In case A is forced to failure, for every positive literal B encountered in the derivation from A to, respectively, A or not A, fact $yesB$ is removed from $\mathcal{T}able(\Pi)$ (if present).*

(3) Literal not A succeeds if one of the following is the case:
  (a) Fact not A is present in $\mathcal{T}able(\Pi)$.

(b) A fails.

(c) A does not fail, rather any derivation of not A incurs through layers of negation again into not A; In this case we say that not A is allowed to succeed.

In consequence of success of not A, fact $yesA$ is removed from $\mathcal{T}able(\Pi)$ (if present), and fact not A is added to $\mathcal{T}able(\Pi)$. In case however not A is allowed to succeed*, in case the parent subgoal fails $yesA$ is restored and not A is removed.*

(4) Literal not A fails if A succeeds.

From this extension of the notions of success and failure we obtain RAS-XSB-Resolution as an extended XSB-Resolution. The "naive" definition is the following (a precise operational definition will require a punctual modification of all definitions related to XSB).

Definition 15 (A "naive" RAS-XSB-resolution). *Given a program Π, let assume as input the data structure $\mathcal{T}able(\Pi)$ used by the proof procedure for tabling purposes, i.e., the table associated with the program (or simply "program table"). Given a query $?\!- A$, the list of current subgoals is initially set to $\mathcal{L}_1 = \{A\}$. If in the construction of a proof-tree for $?\!- A$ a literal L_{i_j} is selected in the list of current subgoals $\mathcal{L}_i$, we have that: if L_{i_j} succeeds then we take L_{i_j} as proved and proceed to prove $L_{i_{j+1}}$ after the related updates to the program table. Otherwise, we have to backtrack to the previous list $\mathcal{L}_{i-1}$ of subgoals. Conditions for success and failure are those specified in Definition 14. Success and failure determine the modifications to $\mathcal{T}able(\Pi)$ specified for XSB-resolution, plus those specified in Definition 14. Backtracking involves restoring previous contents of $\mathcal{T}able(\Pi)$.*

Definition 16. *Given a program Π, its associated table $\mathcal{T}able(\Pi)$, a* free query *is a query $?\!- A$ which is posed on Π when the table has just been initialized. A* contextual query *is a query $?\!- B$ which is posed on Π leaving the associated table in the state created by former queries.*

Success of query $?\!- A$ means that there exist resource-based answer sets that contain A. These sets are further characterized by the final content of $\mathcal{T}able(\Pi)$, which encompasses a number of literals which hold in therein. Backtracking on $?\!- A$ accounts to asking whether there are other different resource-based answer sets containing A, and implies accordingly backtracking $\mathcal{T}able(\Pi)$ to previous contents. Instead, posing a subsequent query $?\!- B$ without resetting the contents of $\mathcal{T}able(\Pi)$, which constitutes a *context*, accounts to asking whether some of the already-computed resource-based answer sets containing A also contain B. Contextual queries and sequences of contextual queries are formally defined below.

Definition 17 (Query sequence). *Given a program Π and $k > 1$ queries $?\!- A_1, \ldots, ?\!- A_k$ performed one after the other, $\mathcal{T}able(\Pi)$ is initialized only before posing $?\!- A_1$. Thus, $?\!- A_1$ is a free query where each $?\!- A_i$, is contextual w.r.t. the previous ones.*

To show the application of RAS-XSB-resolution to single queries and to a query sequence, let us consider the following sample program Π, which includes virtually all cases of potential success and failure. The well-founded model of this program is $\langle\{e\}, \{d\}\rangle$, since e is true as it is a fact, d is false as it has no defining rules, and

all the other atoms are undefined. In fact, they are involved in negative circularities either directly or indirectly (through dependencies, like s and f). There is an even cycle involving a and g, and a unary odd cycle on p, which however depends upon its own negation indirectly, i.e., p depends upon h which in turn depends upon *not* p.

$r_1.\ a \leftarrow \mathit{not}\ g.$ $\quad r_3.\ s \leftarrow \mathit{not}\ p.$ $\quad r_5.\ h \leftarrow \mathit{not}\ p.$ $\quad r_7.\ f \leftarrow \mathit{not}\ g, e.$
$r_2.\ g \leftarrow \mathit{not}\ a.$ $\quad r_4.\ p \leftarrow h.$ $\quad r_6.\ f \leftarrow \mathit{not}\ a, d.$ $\quad r_8.\ e.$

The resource-based answer sets of such Π are $M_1 = \{e, a, f, s\}$ and $M_2 = \{e, s, g\}$.

Below we illustrate some derivations. Initially, $\mathcal{T}able(\Pi)$ includes $yesA$ for every atom occurring in some rule head: $\mathcal{T}able(\Pi) = \{yesa, yesb, yesc, yese, yesf, yesg, yesp, yesh, yess\}$. Let us go through the proof of query ?– f, assuming to adopt a Prolog-like search strategy: applicable rules from first to last as they occur in the program, literals in rule bodies from left to right. Each additional layer of ?– indicates nested derivation of A whenever literal *not* A is encountered. In the comment, we refer to cases of RAS-XSB-resolution as specified in Definition 14.

?– f.
?– *not* a, d. % via r_6
?–?– a.
?–?– *not* g. % via r_1
?–?–?– g.
?–?–?– *not* a. % via r_2. *not* a succeeds by case 3.c, $\mathcal{T}able(\Pi) = \mathcal{T}able(\Pi) \cup \{\mathit{not}\ a\} \setminus \{yesa\}$
?– d. % d fails by case 2b, previous $\mathcal{T}able(\Pi)$ restored, backtracking
?– *not* g, e. % via r_7
?–?– g.
?–?– *not* a. % via r_2
?–?–?– a.
?–?–?– *not* g. % via r_1. *not* g succeeds by case 3.c, a succeeds by case 1.b,
$\mathcal{T}able(\Pi) = \mathcal{T}able(\Pi) \cup \{a, \mathit{not}\ g\} \setminus \{yesa, yesg\}$
?– e. % e succeeds by case 1.b, overall query f succeeds by case 1.b
$\mathcal{T}able(\Pi) = \mathcal{T}able(\Pi) \cup \{e, f\} \setminus \{yese, yesf\}$

Assuming now to go on to query the same context, i.e., without re-initializing $\mathcal{T}able(\Pi)$, queries ?– c and ?– g quickly fail by cases 3.c and 1.a, since $a \in \mathcal{T}able(\Pi)$. Query ?– e succeeds immediately by case 1.a as $e \in \mathcal{T}able(\Pi)$. We can see that the context we are within corresponds to resource-based answer set M_1, where only s remains to be proved. This can be done as follows:

?– s.
?– *not* p. % via r_3
?–?– p.
?–?– h. % via r_4
?–?– *not* p. % via r_5, p fails by case 2.c.iii, h fails by case 2.c.ii, $\mathcal{T}able(\Pi) =$
$\mathcal{T}able(\Pi) \setminus \{yesp, yesh\}$. *not* p succeeds by case 3.b, s succeeds by case 1.b,
$\mathcal{T}able(\Pi) = \mathcal{T}able(\Pi) \cup \{\mathit{not}\ p, s\} \setminus \{yess\}$

It remains to show how the derivation of h proceeds, as it involves the tricky case of a positive dependency through negation, where h is still undefined under the well-founded semantics.

?– h.
?– *not* p. % via r_5
?–?– p.

?–?– h. % h fails by case 2.c.iv. *not* p succeeds by case 3.b
$\mathcal{T}able(\Pi) = \mathcal{T}able(\Pi) \setminus \{yesp, yesh\} \cup \{\textit{not } p\}$

6 Properties of RAS-XSB-resolution

Properties of resource-based answer set semantics are strictly related to properties of RAS-XSB-resolution. In fact, thanks to Relevance we can have soundness and correctness, and Modularity allows for contextual query and locality in constraint-checking.

Theorem 1. *RAS-XSB-resolution is correct and complete w.r.t. resource Answer Set semantics, in the sense that, given program Π, query ?– A succeeds under RAS-XSB-resolution with an initialized $\mathcal{T}able(\Pi)$ iff there exists resource-based answer set M for Π where $A \in M$.*

Theorem 2. *RAS-XSB-resolution is contextually correct and complete w.r.t. resource Answer Set semantics, in the sense that, given program Π and query sequence ?– A_1, …, ?– A_k, $k > 1$, we have that, for $\{B_1, \ldots, B_r\} \subseteq \{A_1, \ldots, A_k\}$ and $\{D_1, \ldots, D_s\} \subseteq \{A_1, \ldots, A_k\}$, the queries ?– B_1, …, ?– B_r succeed while ?– D_1, …, ?– D_s fail under RAS-XSB-resolution, iff there exists resource-based answer set M for Π where $\{B_1, \ldots, B_r\} \subseteq M$ and $\{D_1, \ldots, D_s\} \cap M = \emptyset$.*

In fact, keeping in $\mathcal{T}able(\Pi)$ atoms and literals proved so far accounts to performing the simplification of given program Π w.r.t. a resource-based answer set computed for the subprogram including relevant rules of previous queries. Therefore, the result descends from Modularity of resource-based answer set semantics. It remains to consider the issue of constraint checking. Notice that, due to modularity of RAS, if Π is admissible, then only constraints relevant to given query need to be checked.

Proposition 4. *Let Π be an admissible program w.r.t. the constraints $\leftarrow H_1, \ldots, \leftarrow H_h$ (it has admissible answer sets). Let $\leftarrow H_1, \ldots, \leftarrow H_k$, $k \leq h$ be the relevant constraints for a query ?–A. Then, if ?–A succeeds and each H_i, $i \leq k$, considered as a query, contextually succeeds as well, then there exists some admissible resource-based answer set M for Π with $A \in M$.*

If admissibility of Π is unknown, all constraints must instead be checked. Checking constraints on the state of $\mathcal{T}able(\Pi)$ left by a query alleviates the efficiency problem. "Smart" heuristics, such as those presently adopted by answer set solvers, for checking constraints during the proof process might also be in order.

7 Discussion and Concluding Remarks

A relevant question about RAS-XSB-resolution concerns whether it is applicable to non-ground queries and programs. By resorting to standard unification, non-ground queries on ground programs are managed without substantial modifications. We claim that the procedure can be extended to non-ground programs without requiring preliminary program grounding. Transforming this claim into evidence requires however an actual reworking of XSB-resolution definitions and proofs. This is a topic for future work.

Another relevant question is whether RAS-XSB-resolution might be extended to plain ASP. Unfortunately, an ASP program may have a quite complicated structure: the effort of in [7] has been in fact that of performing a layer-based computation upon some conditions. Thus, the adoption of RAS-XSB-resolution is possible at the condition of structuring an ASP program so that constraints are at the top layer. Many applications are already expressed in this form, which means that the proposed procedure may have an impact beyond resource-based answer set semantics.

In summary, we have proposed the theoretical foundations of a proof procedure related to a reasonable extension of answer set programming. The procedure has been obtained by exploiting properties of both answer set semantics and resource-based answer set semantics, which enable us to resort as a starting point to XSB-resolution. The new procedure has drawn inspiration from the tabling feature of XSB-resolution. Future work includes a precise definition of RAS-XSB-resolution, and an implementation, that should then be checked and experimented on (suitable versions of) well-established benchmarks (see, e.g., [34, 35]). We also intend to investigate an integration of RAS-XSB-resolution with principle and techniques proposed in [8], so as to further enlarge its applicability.

References

[1] Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K., eds.: 5th ICSLP, MIT Press (1988) 1070–1080

[2] Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385

[3] Marek, V.W., Truszczyński, M. In: Stable logic programming - an alternative logic programming paradigm. Springer (1999) 375–398

[4] Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press (2003)

[5] Truszczyński, M.: Logic programming for knowledge representation. In Dahl, V., Niemelä, I., eds.: Logic Programming, 23rd Intl. Conference, ICLP 2007. (2007) 76–88

[6] Gelfond, M.: Answer sets. In: Handbook of Knowledge Representation. Chapter 7. Elsevier (2007)

[7] Gebser, M., Gharib, M., Mercer, R.E., Schaub, T.: Monotonic answer set programming. J. Log. Comput. **19**(4) (2009) 539–564

[8] Bonatti, P.A., Pontelli, E., Son, T.C.: Credulous resolution for answer set programming. In Fox, D., Gomes, C.P., eds.: AAAI 2008, AAAI Press (2008) 418–423

[9] Web references on ASP: Clasp: `potassco.sourceforge.net`; Cmodels: `www.cs.utexas.edu/users/tag/cmodels`; DLV: `www.dbai.tuwien.ac.at/proj/dlv`; Smodels: `www.tcs.hut.fi/Software/smodels`.

[10] Dix, J.: A classification theory of semantics of normal logic programs I-II. Fundam. Inform. **22**(3) (1995) 227–255 and 257–288

[11] Costantini, S., Formisano, A.: Answer set programming with resources. Journal of Logic and Computation **20**(2) (2010) 533–571

[12] Costantini, S., Formisano, A.: Modeling preferences and conditional preferences on resource consumption and production in ASP. Journal of Algorithms in Cognition, Informatics and Logic **64**(1) (2009) 3–15

[13] Costantini, S., Formisano, A., Petturiti, D.: Extending and implementing RASP. Fundamenta Informaticae **105**(1-2) (2010) 1–33

[14] Costantini, S., Formisano, A.: RASP and ASP as a fragment of linear logic. Journal of Applied Non-Classical Logics (JANCL) **23**(1-2) (2013) 49–74
[15] Girard, J.Y.: Linear logic. Theoretical Computer Science **50** (1987) 1–102
[16] Costantini, S., Formisano, A.: Negation as a resource: A novel view on answer set semantics. In Cabalar, P., Son, T.C., eds.: LPNMR 2013. Vol. 8148 of LNCS., Springer (2013) 257–263 Long Version in [36].
[17] Marek, V.W., Truszczynski, M.: Reflective autoepistemic logic and logic programming. In: LPNMR. (1993) 115–131
[18] Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM **38**(3) (1991) 620–650
[19] Swift, T., Warren, D.S.: Xsb: Extending Prolog with tabled logic programming. TPLP **12**(1-2) (2012) 157–187
[20] Chen, W., Warren, D.S.: A goal-oriented approach to computing the well-founded semantics. J. Log. Program. **17**(2/3&4) (1993) 279–300
[21] Costantini, S., Formisano, A.: Query answering in resource-based answer set semantics. Extended version, available at `http://www.dmi.unipg.it/formis/papers/CosForASPOCPExtended.pdf`
[22] Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag (1987)
[23] Apt, K.R., Bol, R.N.: Logic programming and negation: A survey. J. Log. Program. **19/20** (1994) 9–71
[24] Costantini, S.: Contributions to the stable model semantics of logic programs with negation. Theoretical Computer Science **149**(2) (1995) 231–255
[25] Costantini, S.: On the existence of stable models of non-stratified logic programs. Theory and Practice of Logic Programming **6**(1-2) (2006)
[26] Lin, F., Zhao, X.: On odd and even cycles in normal logic programs. In McGuinness, D.L., Ferguson, G., eds.: Proceedings of AAAI 2004, AAAI Press / The MIT Press (2004) 80–85
[27] Costantini, S., Formisano, A.: Resource-based answer set semantics. Submitted to a journal, draft available at `www.dmi.unipg.it/formis/papers/CF_NARdraft.pdf`
[28] Pereira, L.M., Pinto, A.M.: Revised stable models - a semantics for logic programs. In Bento, C., Cardoso, A., Dias, G., eds.: Progress in Artificial Intelligence, Proc. of EPIA 2005. Vol. 3808 of LNCS., Springer (2005)
[29] Pereira, L.M., Pinto, A.M.: Tight semantics for logic programs. In Hermenegildo, M.V., Schaub, T., eds.: Tech. Comm. ICLP 2010. Vol. 7 of LIPIcs. (2010) 134–143
[30] Osorio, M., López, A.: Expressing the stable semantics in terms of the pstable semantics. In: Proc. of the LoLaCOM06 Workshop. Vol. 220 of CEUR Workshop Proc. . (2006)
[31] Osorio, M., Pérez, J.A.N., Ramírez, J.R.A., Macías, V.B.: Logics with common weak completions. J. Log. Comput. **16**(6) (2006) 867–890
[32] Lifschitz, V., Turner, H.: Splitting a logic program. In: Proc. of ICLP'94. (1994) 23–37
[33] Cadoli, M.: The complexity of model checking for circumscriptive formulae. Inf. Process. Lett. **44**(3) (1992) 113–118
[34] Calimeri, F., Ianni, G., Krennwallner, T., Ricca, F.: The answer set programming competition. AI Magazine **33**(4) (2012) 114–118
[35] Alviano, M. et al.: The fourth answer set programming competition: Preliminary report. In Cabalar, P., Son, T.C., eds.: Proc. of LPNMR 2013. Vol. 8148 of LNCS. (2013) 42–53
[36] Costantini, S., Formisano, A.: Negation as a resource: A novel view on answer set semantics. In Cantone, D., Nicolosi Asmundo, M., eds.: CILC 2013. Vol. 1068 of CEUR Workshop Proceedings. (2013)